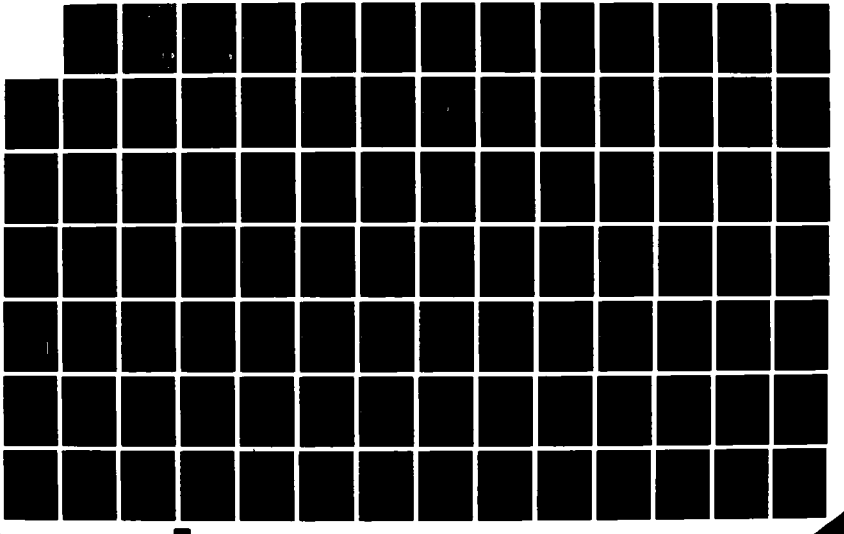
AD-A192 833 IMAGE PROCESSING USING A PARALLEL ARCHITECTURE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING B R HODGES DEC 87 AFIT/GE/ENG/87D-25 1/2

UNCLASSIFIED F/G 12/9 NL



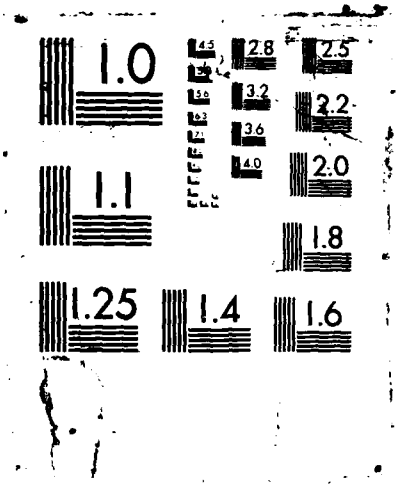

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

IMAGE PROCESSING USING A PARALLEL ARCHITECTURE

THESIS

Billy R. Hodges
Captain, USAF

AFIT/GE/ENG/87D-25

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/87D-25

IMAGE PROCESSING USING A PARALLEL ARCHITECTURE

THESIS

Billy R. Hodges
Captain, USAF

AFIT/GE/ENG/87D-25

Approved for public release; distribution unlimited

AFIT/GE/ENG/87D-25

IMAGE PROCESSING USING A PARALLEL ARCHITECTURE

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

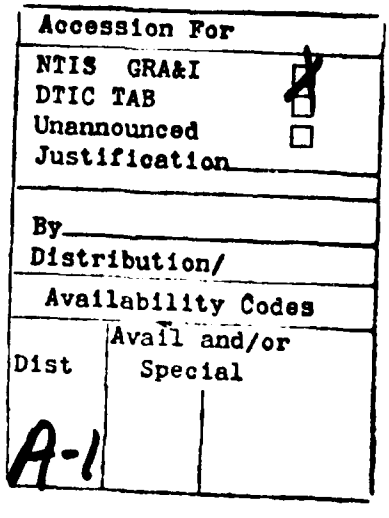
Requirements for the Degree of

Master of Science in Electrical Engineering

Billy R. Hodges, B.S.E.E.

Captain, USAF

December 1987



Approved for public release; distribution unlimited

Acknowledgements

First and foremost, I thank the Lord for giving me the ability and the opportunity to undertake this thesis effort. It has been a difficult time, but He has used it to strengthen me both mentally and spiritually. I also wish to express my thanks to Lt Col Walter Seward for placing before me worthy goals for my thesis and for my career as an officer. I am also gratefully to Maj Glenn Prescott and Dr. Gary Lamont for their assistance. A special thanks goes to Dr. Mathew Kabrisky for sharing his time and experience. And finally, I thank my helpmate for being so.

Bill Hodges

Table of Contents

List of Figures

Abstract

This study developed a set of low level image processing tools on a parallel computer that allows concurrent processing of images in order to support development of systems that use multiple images to gather information. The parallel computer used is a collection of powerful processors connected in a hypercube topology.

The software developed simplifies the interface between the parallel computer and the applications developer by providing a library of functions in the C programming language. These functions are used to control "image processor programs" that run independently in the parallel computer and perform the image processing operations. The complexities of the parallel processor are hidden and replaced with a flexible structure specifically designed for image processing. This structure provides a simplified interface, but also acts as a framework to which additional image processing operations can be added.

Timing measurements indicate that, in addition to providing a unique applications development environment, the set of tools offers a significant reduction in the time required to perform some commonly used image processing operations.

IMAGE PROCESSING USING A PARALLEL ARCHITECTURE

I. Introduction

This thesis was prompted by the growing need for sophisticated computer systems that can process large amounts of image information in a limited amount of time. Developing these systems will require new methodologies and equipment, and as a first step toward developing these systems, a structured set of image processing tools was implemented using a parallel computer. More important than potential speed increases, a parallel computer offers a whole new development environment not possible with uni-processor systems. Parallel computers avail researchers of a way to create new techniques and a vehicle to implement these techniques. It is intended that the tool set be used for image analysis, expanded to include other tools, and in the future be implemented as a part of a real-time image processing system.

Background

As man has developed complex machines, more information is needed for the machines' operation in control and decision-making roles. In the past, information has been input to the machines by human operators, but this is not always a desirable solution. Humans can not react quickly

enough in some situations or are subject to fatigue and loss of concentration. Also, some machines operate in environments where human presence is either not desirable or not possible. An alternative to the "human in the link" approach is to devise better techniques that use machine sensor sets to gather the information needed for the machine's operation. This will eliminate the need for a human, using his sensor set, to provide information to the machine.

It is said that humans receive as much as 75% of their information about the world around them through vision (10:1) with the rest coming from the other senses: hearing, smell, touch, and taste. Given this vast amount of information humans gather through their eyes, much research has been dedicated to developing the capability for machines to analyze images and obtain the same information as humans. It is the hypothesis that there exists a feature set contained within a scene that, if identified and isolated, can be used by a machine to gather what is considered the important information. While it is the intent to allow machines to use image data, this does not necessarily imply that the methods used in "machine vision" need be based on a model of the human vision mechanism or, furthermore, even remotely resemble the human visual system. It is true that an understanding of the human visual system can aid in the development of machine vision systems, but humans and

machines are vastly different entities; what is meaningful to the human system may not be useful at all to a machine system. One could attempt to duplicate the human vision system in order to produce a quality machine vision system but, this seems a difficult way to solve the problem. An alternative is to find a method that will reduce a scene to terms that can be meaningful in a model that is more readily implementable in a machine. Another important difference between the human system and the machine system is the type of sensors available to each: the human system has the visual light range and the other four senses available while the machine can make use of other ranges of light, in particular infared, as well as numerous other passive and active sensors just as easily as it can use the visual range. It is the use of multiple sensors that seems to hold the most promise for machine vision systems.

One of the areas of machine vision research is digital image processing. Here a scene is encoded in a digital computer format, binary numbers, and then manipulated by the use of special purpose computer hardware controlled by firmware or, as in much research, by the use of a general purpose digital computer. The digital computer allows great flexibility in the design and implementation of machine vision systems. Also, image processing techniques that are difficult, expensive, or not possible by other means, such as optical or analog processing, can be implemented using a digital computer. The history of computer image processing

is relatively brief with many of the techniques being developed, exploited, or applied in the last two decades (10:1).

There are already a number of fields that have benefited from machine vision in general and, of particular interest, from digital image processing. In manufacturing, machine vision has helped to improve some manufacturing processes and increase product safety and reliability (12:15). One of the simplest examples of machine vision is a photoelectric machine guard that uses a set of light sources and light receivers to detect a human operator's stray limbs and stops the operation of the machinery when any of the light paths are interrupted (12:3). Another example, which uses more complex techniques, is a system that scans a conveyor belt to identify metal castings as one of six types and passes the type and position information to a set of robot arms used to sort the castings into skips (12:5-6). Other systems that use machine vision in manufacturing include crack detection in glass and metal parts, detection of skewed labels on product containers, detection of blemished food products, and inspection of sheet metal, photographic film, and other products for surface defects (12:18-20). In the integrated circuit (IC) industry machine vision is used in chip assembly and defect detection (14:12). Also, there is a die-bond machine which uses machine vision to isolate bonding pads and connects select pairs of these pads with wire (14:12).

In military applications, image interpretation is complicated by several factors. First, conditions such as lighting and perspective can not be controlled as in commercial applications. Second, it is often the case that the objects that are being sought are purposefully hidden. Thus, while techniques to successfully find and identify a metal casting on a conveyer belt exist, techniques to find tanks in a forest are proving more difficult. However, there are a number of weapon system concepts that are being developed that rely on image processing as a primary mode of gathering information.

It has been proposed that the accuracy of strategic and conventional missiles can be improved by supplementing inertial systems with a guidance system that matches imagery along the flight-path and in the terminal area with a stored database (22). Another autonomous target recognition (ATR) technique that is being studied is using laser rangefinder information to guide air-to-surface missiles to their targets (7). A proposed guidance technique for anti-tank munitions uses the large infared signatures of the tank's exhaust system to identify targets in a scene (26). Another notable system that is under development for fighter aircraft is the LANTIRN system which is designed to provide navigation and targeting capabilities in adverse conditions using forward looking infared (FLIR) sensors. It is projected that future guidance systems will require multiple sensors to deal with the more complex battlefield of tomorrow

(8:221). Approaches are being developed to integrate or "fuse" information from electro-optical, FLIR, and millimeter-wave (MMW) radar. The range of conditions under which a system can perform may be extended by this combination of sensors. For instance, one sensor may be able to provide useful information when another can not, or it may be possible to blend information from two sensors to achieve a symbiotic effect. These new approaches are required for the detection and identification of strategic targets, a task which has been complicated by the increased mobility of those targets and by camouflage, concealment, and deception (CCD) techniques employed to increase their survivability (28:180).

One of the most serious drawbacks to digital image processing (or digital computation in general) is the amount of time required to process increasingly large amounts of data. Using a single processor each picture element (or pixel) of the entire image array must be operated on in turn. The system performance trade-off then becomes one of a greater numbers of pixels, giving more detail, for a longer time required to complete the processing.

A complicating factor is, in many cases, the time required to execute an algorithm does not grow linearly with the number of pixels considered but rather, grows proportional to the square or cube of the number of pixels. This provides an even greater impetus to find methods which increase the computational speed available in computers.

One method to decrease the required processing time is the use of multiple computers, as opposed to the traditional method of using a single computer. By using multi-processor computer systems, a computational task can be divided into its independent sub-tasks which can be performed simultaneously thereby increasing the computational capability applied to the task.

Thesis Problem

The AFIT Department of Electrical and Computer Engineering has recently acquired two Intel iPSC hypercubes, and is interested in developing an image processing capability on these machines. There are numerous algorithms and operations used in image processing whose execution time can be significantly decreased by using these multi-processor computers. This thesis effort resulted in the development of a low-level image processing capability for the iPSC hypercubes. Low-level operations used in image processing include image retrieval and storage, image copying and movement with the processing structure, elementary mathematical functions, and other common operations used in image analysis including correlation. The software developed was designed to fit into a larger hierarchical structure which is presented in Chapter 3. This is to allow further expansion at the lowest level as well as the development of higher tiers in the hierarchy. This multi-level approach

allows for an orderly growth path for the project in further thesis efforts.

Scope

The goal of this thesis effort was to develop a fundamental image processing software package for a parallel computer. Further, this package should allow, not only the use of multiple processors for faster image processing, but also allow multiple tasks to be performed concurrently. Examples of these concurrent tasks include processing data from two or more sensors (multi-sensor fusion) and processing multiple images from one sensor separated in time (temporal processing).

The image processing algorithms implemented were chosen from the set of those most commonly used in the field. The objective was not to develop image processing techniques but, to develop a tool with which image processing research can be conducted. The hypercube was used in its native configuration. That is, no attempt was made to map it into another architecture such as a tree, mesh, or grid. Particular attention was given to the view that the software developed is to be used for any of a number of future projects and, as such, flexibility is most important.

At the present, machine vision and image processing are in the beginning stages and consequently very few techniques beyond the lowest level are widely established in the field. Because this is the case, sufficient documentation is

included to promote further development of the package while maintaining the structure needed for a multi-processor, multi-task tool.

Approach

The first task was to develop the software system structure for the the project. Second, a 2 dimensional Fast Fourier Transform (2D-FFT) routine written by Intel was incorporated into this structure. Next, an image display routine was developed on a Sun Microsystems workstation so that the input and resultant images could be displayed. Then, the bulk of the image processing routines were written on the hypercube and finally, a portion of the image processing routines was moved to run on the Sun workstation rather than the Intel host computer by using a "remote-hosting" software package written by Oakridge National Labs.

Materials and Equipment

Two Intel iPSC hypercube computers were used to develop the image processing routines. The hypercubes consist of two primary parts: the host computer and the cube. The host computer is a Xenix based multi-user, multitasking computer. It is used to edit and compile all software for it and the cube. The cube is a collection of 2^N processors, where N is 1 to 7, arranged in a hypercube configuration which, under the present operating system, is viewed as a system resource that is allocated to one user at a time.

A Sun Microsystems workstation is used to run the host software that activates the cube image processing routines and to display the images. The Sun workstation has excellent graphics capability, a multi-window environment and runs the Unix operating system. This combination makes it an excellent vehicle for future projects ranging from an interactive image processing workstation to an expert system based image analysis package. In each application the hypercube can be used to run multiple computationally intensive tasks with software on the Sun providing display, control, and decision-making or supervisory capabilities.

Presentation

Chapter 2 presents an explanation of the image processing algorithms implemented along with some introductory material on parallel processing. The next chapter presents the system requirements, design analysis, and a system definition. The fourth chapter outlines the detailed design, coding, and testing of the software. Chapter 5 presents an analysis of the effectiveness of the software developed and the last chapter covers some conclusions and recommendations for further study. The Appendix is a user's manual that details how to use the image processing software developed during this thesis.

II. Review of the Literature

Overview

A potential way of decreasing the amount of time required to complete a processing task is through the use of parallel processing. As mentioned in Chapter 1, image processing is one of the applications that can benefit from this emerging technology.

The fundamentals of parallel processors are presented here to provide a background for programming a parallel processor. Next, a general introduction to the field of image processing is given including a development of the algorithms selected to be implemented.

Parallel Processors

Since the first electronic computer, ENIAC, was built in 1946 typical electronic computers have had the basic structure shown in Figure 1. This architecture, known as the von Neumann architecture, is described by Baer (3:4) as consisting of the following five basic parts: 1) the input receives data and instructions from the user, 2) the memory stores the instructions (program), data, and any results, 3) the arithmetic and logic unit (ALU) performs all arithmetical and logic operations on the data, 4) the control unit retrieves and executes instructions from the memory and 5) the output transmits resulting data to the user. As a program is running, the control unit fetches the next

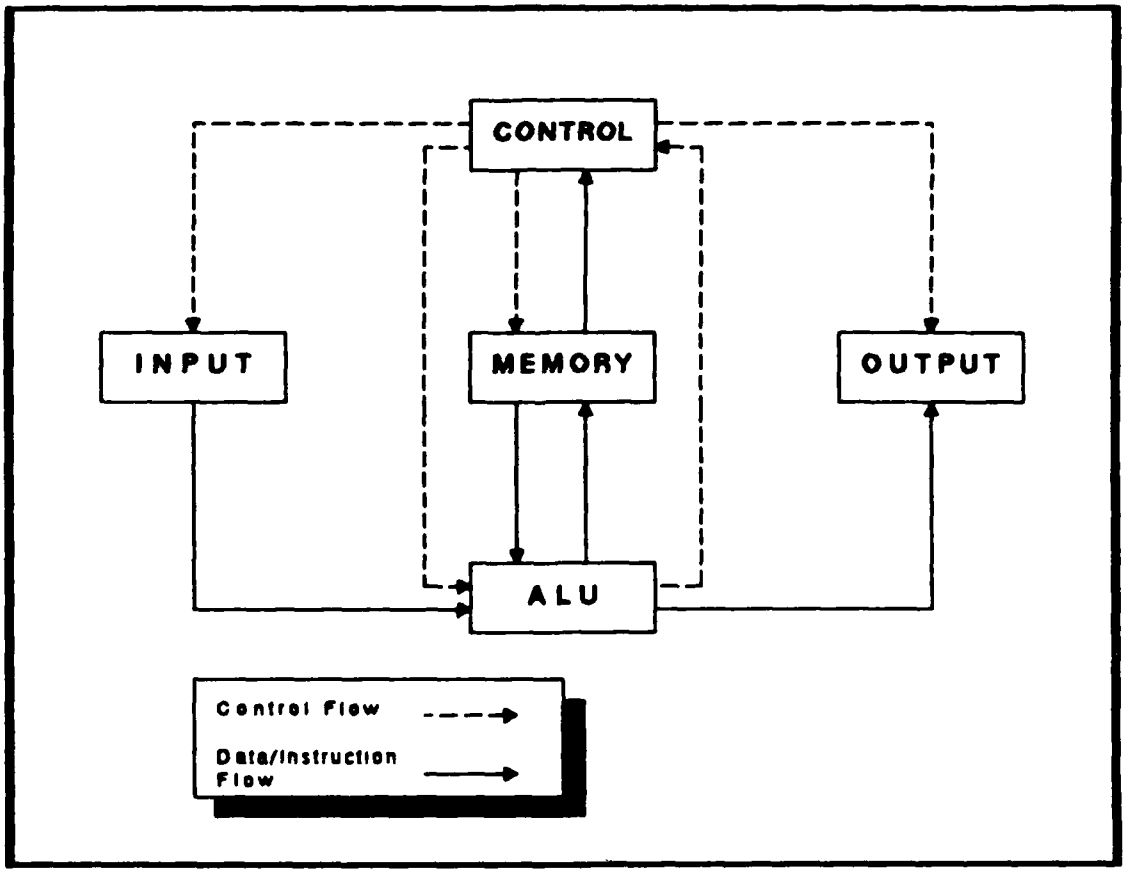


Figure 1. Basic Computer Architecture (3:5)

instruction from memory, decodes the instruction, and then either activates the input, the output, or the ALU. This cycle continues until the program is complete.

It has been estimated that the maximum theoretical computation rate for computers built around this single processor arrangement is about 3 billion floating point operations per second (3 Giga-FLOPS) (27:46). This limit is imposed by the laws of physics which state that information can not be transmitted faster than the speed of light. In practice, today's peak computation rates for these

types of computers are two or three orders of magnitude less. While there is still much processing capability still to be gained in theory, practical limits such as impurities in materials are hampering further increases. If multiple copies of these single processors were coordinated to work on parts of a computation in parallel, a reduction in processing time over that of a single processor computation time can, in many cases, be achieved.

The term "parallel processor" is used to describe a number of different computer architectures that contain more than one processor. Baer (3:491-494) presents Flynn's taxonomy as one way of classifying the various computer architectures.

Flynn's Taxonomy. Around 1966 Flynn devised a method of classifying computer architectures based upon number of data streams and the number of instruction streams. Each of these is further classified as either single or multiple. Thus, there are four categories under Flynn's classification scheme:

Single Instruction	Single Data	(SISD)
Single Instruction	Multiple Data	(SIMD)
Multiple Instruction	Single Data	(MISD)
Multiple Instruction	Multiple Data	(MIMD)

Single Instruction-Single Data. Most computers used today are categorized as SISD. Here a single instruction stream is used by the processor to operate on a data set.

Single Instruction-Multiple Data. An SIMD computer is composed of a number of processors and a controlling unit. The controlling unit feeds instructions to all the processors and they execute the instructions on their respective data sets at the same time. Thus, all processors are synchronized (executing in a lock-step fashion), performing the same operations on their data. An example of an SIMD computer is an array processor, which performs simultaneous arithmetic operations on one-dimensional vectors that can have thousands of elements. This type of parallel processing is used in the Cray supercomputers.

Multiple Instruction-Single Data. Presently, there are few examples of this architecture. Here multiple processors apply differing instructions on the same data stream.

Multiple Instruction-Multiple Data. The MIMD architecture differs from the SIMD in that each processor operates asynchronous to all other processors. Each processor has its own instruction stream and may perform different tasks than all of its neighboring processors.

MIMD Architectures. There are several characteristics that help to further distinguish parallel computers in the MIMD category.

Memory. The first distinguishing characteristic among MIMD machines is whether all processors share a common memory or if each of the processors has its own memory (see Figure 2). The "shared-memory" computer has the advantage

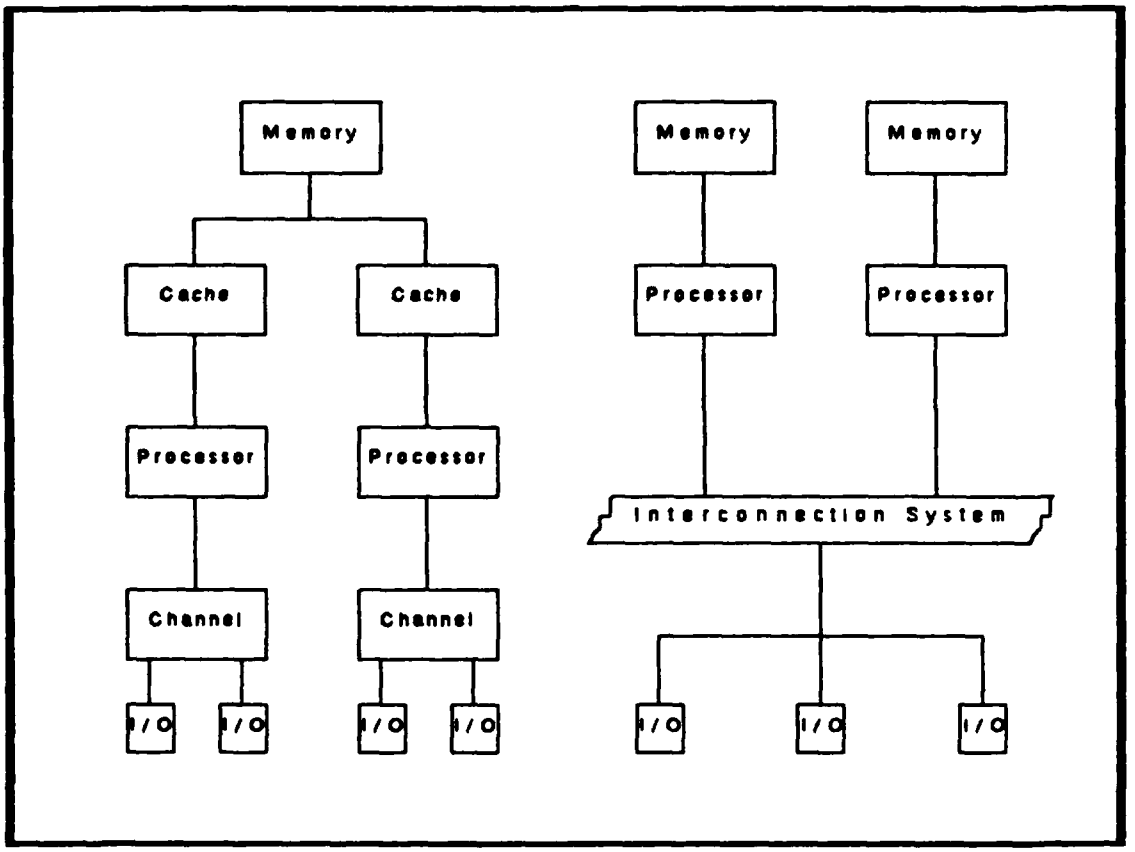


Figure 2. Shared and Non-shared Memory Systems (1:32)

that data commonly used by all processors is available in one place. This saves storage space by avoiding replication as in multiple memory systems and it also simplifies the problem of insuring that all processors have access to the most recent values of the data. These systems are referred to as "tightly coupled" because of their high degree of interdependency. The shared memory system does have some serious drawbacks. First is the problem of devising a method to allow the multiple processors access to the same memory. While dual-port memories do exist, they are significantly

more expensive than memories used in the non-shared memory computers and even then only allow access for two processors. For larger numbers of processors a memory access multiplexing system must be used and this results in contention for access to the memory. This becomes a severe bottleneck as the number of processors accessing the memory increases (17:193).

In contrast to the shared memory computers, non-shared memory computers are referred to as "loosely coupled"; all processors have a local memory and communicate with each other and the outside world via an interconnection system. In these systems the memory access question, associated with shared-memory systems, is traded for a communications question. The system will use "messages" to communicate data from one processor to another through the interconnection system. The number of processors plays an important role in determining the structure of this arrangement.

Processors. Another issue in the implementation of a parallel computer is the number of processors used. Some parallel computers use thousands of simple processors each capable of a small number operations while others use a smaller number of more powerful processors. The ideal situation would be to have a larger number of the more powerful processors, which has to date has proven economically unattractive. The trade-off of numbers verses capability gives rise to the concept of granularity.

> This can be thought of as either a measure of the nominal length of messages sent from processor to processor during execution of a problem, or the number of instructions typically assigned to a processor in a single procedure. Grain size is important, in that fine grain applications for example, fit naturally on a system of many small, less capable processors with very fast communications paths between processors geared to relatively short messages. Course grain applications, those with thousands of instructions per procedure and large interprocessor messages, would fit well on a system of a few large processors and an interconnection medium designed to transfer large blocks of data [1:38].

Once the number of processors has been fixed, the way processors are inter-connected, the topology, must be established.

Topology. Finally, communications links must be provided for data exchange in the non-shared memory MIMD computer. The ideal case would be a complete interconnection (see Figure 3) between all processors or "nodes". In this interconnection topology each processor has a dedicated line to all other processors. No delays are caused by the unavailability of communications channels if each line is bidirectional but, this comes at a cost. Each node must have the memory and processing capability to receive and store information from all its communications channels until the data is needed. Also, the number of lines required for the complete inter-connected topology of N nodes is N(N-1)/2. For a relatively small network of 16 nodes, 120 bidirectional data channels are needed. Therefore, for all but the smallest numbers of nodes the complete interconnect is not a feasible solution. An alternative, is a bus oriented

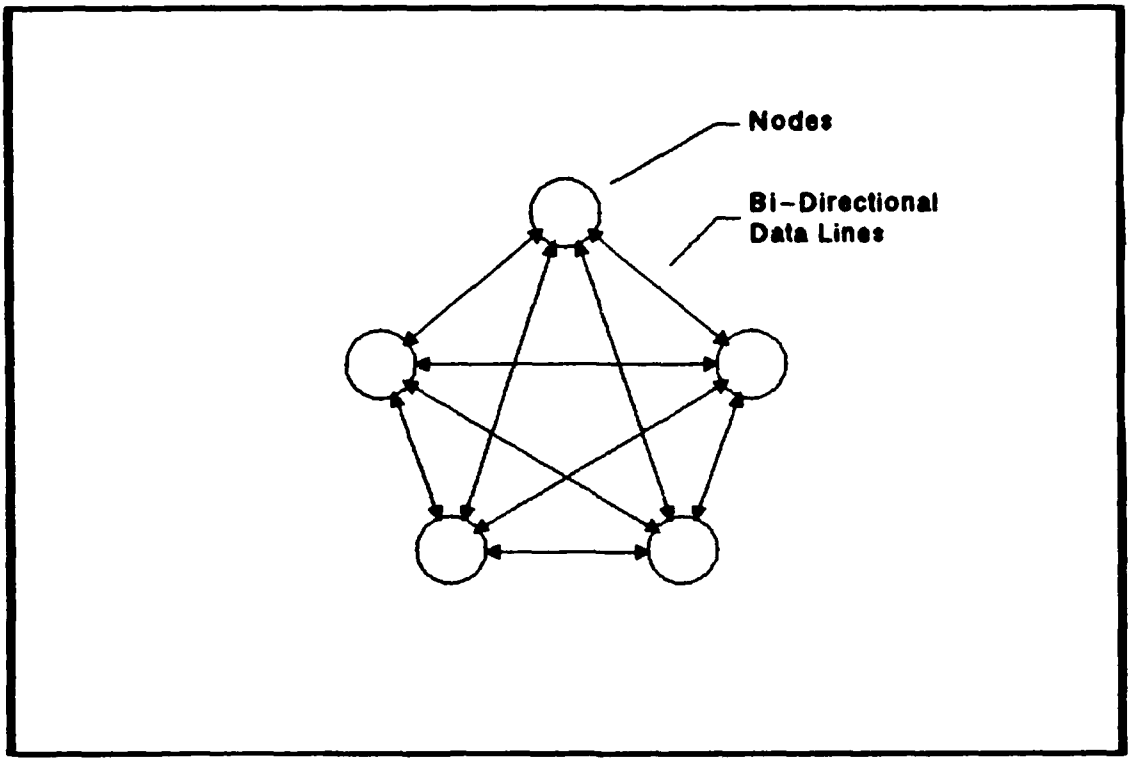


Figure 3. Complete Inter-connect Topology

architecture where all nodes are connected to a single high-bandwidth communications channel (see Figure 4). This topology can be expanded to include large numbers of processors without the rapid growth of data paths associated with the complete inter-connect. However, as the number of nodes grows the data bus may become very congested with data traffic. So, both the complete interconnect and the uni-bus architectures are limited in their usefulness to relatively small networks of nodes.

There are numerous other multi-processor organizations including mesh, pyramid, shuffle-exchange network, butterfly, hypercube (cube-connected), and cube-connected cycles

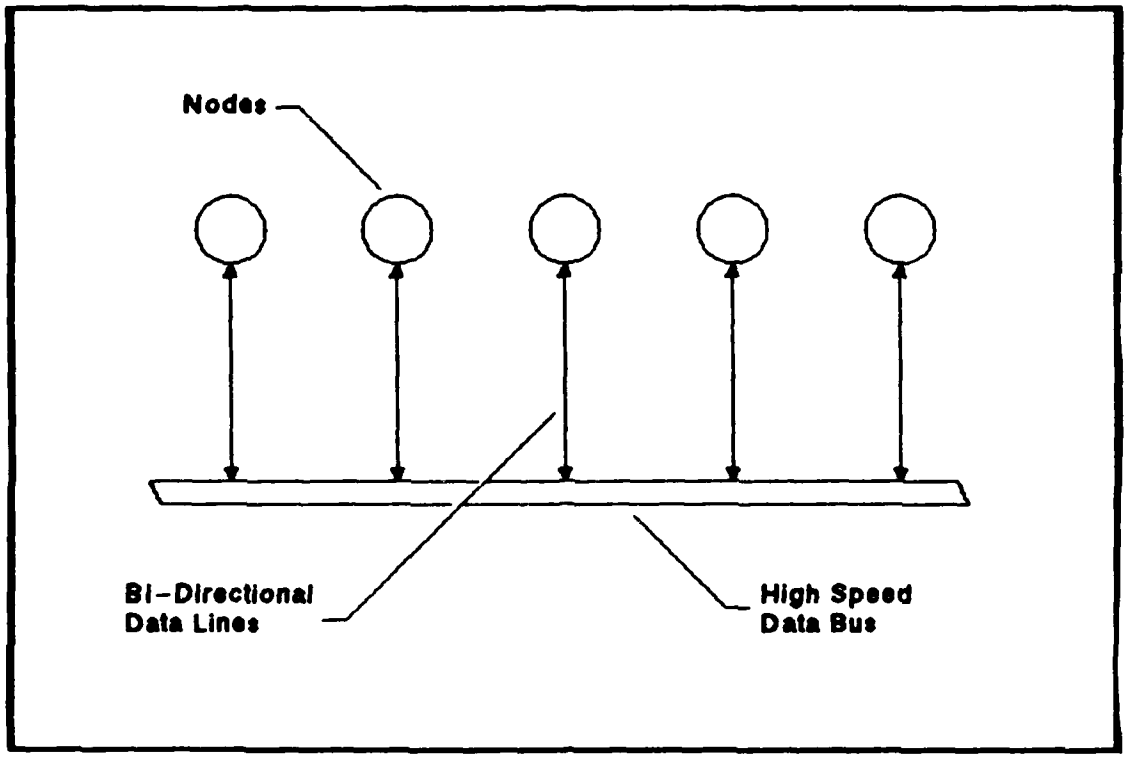


Figure 4. Bus Oriented Topology

(21:25-29). Each of these structures differs in the interconnection scheme used in providing inter-node communications and have different strengths. Some of these are illustrated in Figure 5.

Hypercube. "One of the most powerful interconnection methods is called the hypercube" (17:193). The hypercube has been chosen by several computer manufactures as the interconnection scheme for their commercially available computers. These manufactures include Intel, Amatek, N Cube, TMI, and FPS (9:6). The number of processors or nodes in a hypercube is often denoted by its "dimension." A hypercube said to be of dimension N has 2^N processors. In

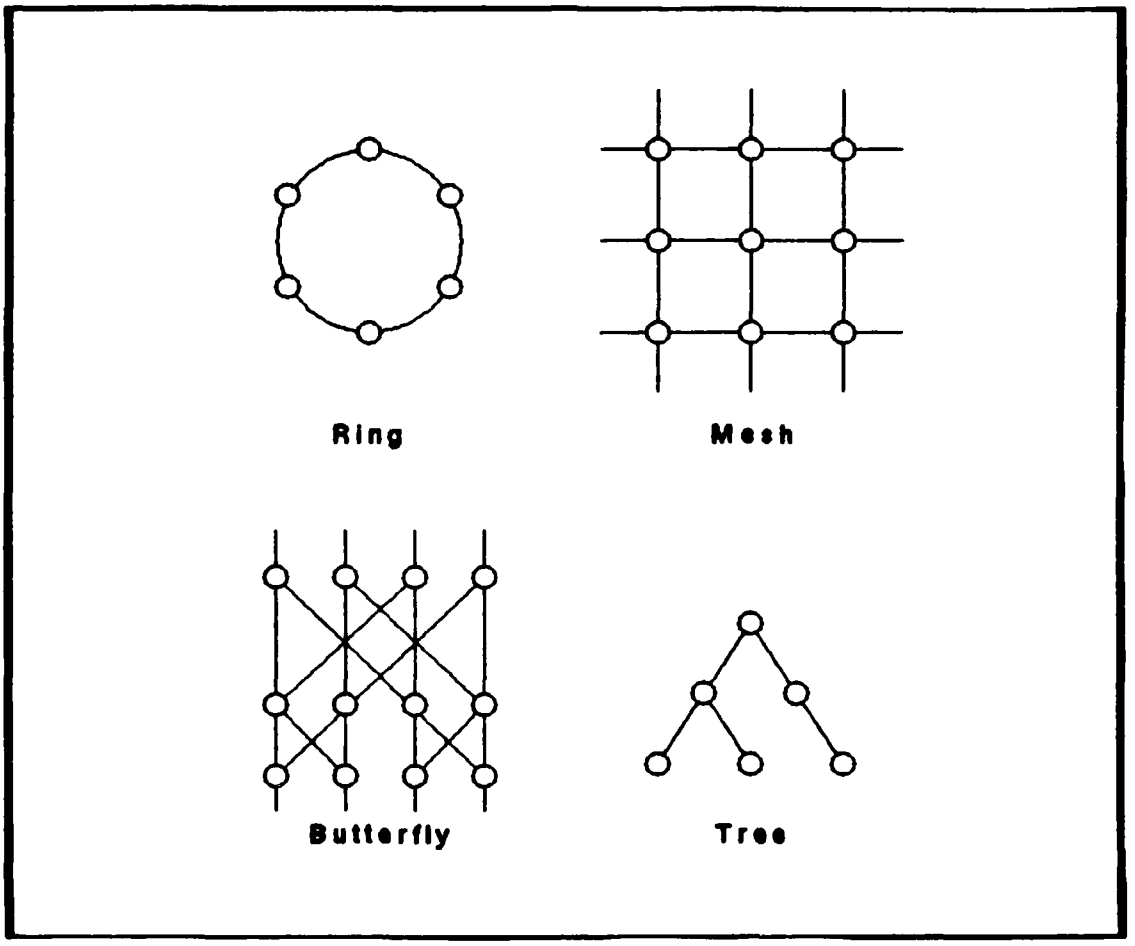


Figure 5. Various Inter-connection Topologies

constructing a cube of dimension N, each of the processors in connected to N other processors. Figure 6 illustrates hypercubes of dimensions 0, 1, 2, 3, and 4. The connections are assigned in the following manner: the nodes are assigned labels equal to the binary representation of the numbers 0 through 2^N-1 and every node is connected to every other node whose binary label differs in one bit position (17:193). Thus, in a dimension 3 cube, the node labeled 100 is connected to nodes 000, 110, and 101. Palmer

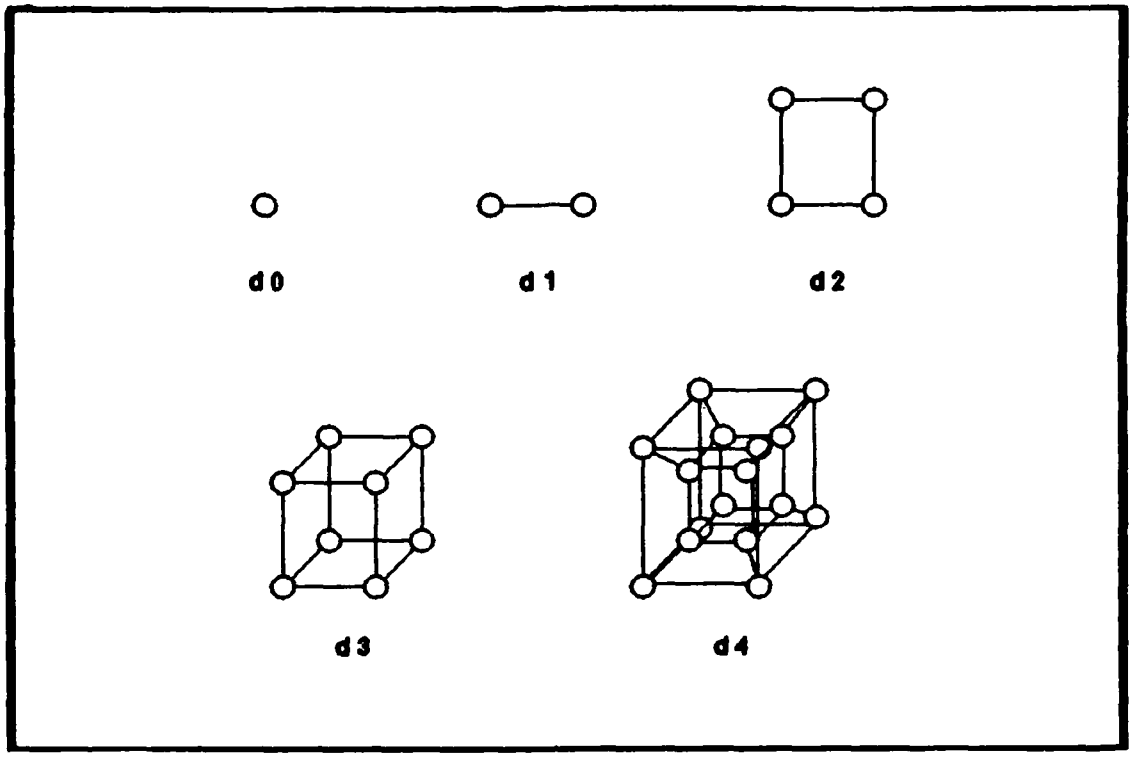


Figure 6. Hypercubes of Various Dimensions

points out the following characteristics the of hypercube:

1) The number of directly connected neighbors to a processors grows with the order of the hypercube. Thus, a hypercube is more densely connected than a mesh or tree.
2) The hypercube can be mapped onto most other useful interconnection schemes.
3) The hypercube is recursive. A hypercube of order N is made of two hypercubes of order N-1.

The first property of hypercubes allows a programmer to ignore the optimum interconnection scheme for a given problem and simply assume that his machine is maximally interconnected. The second property, ... allows a programmer to use almost any logical interconnection that seems optimal for a given problem. And the third property allows programs to be written so that they can run on any order hypercube. Thus, the order of the hypercube can be a parameter of the program. A corollary to this is that an operating system that manages a hypercube array can treat it as a large set of logical hypercubes that can be allocated in sizes requested by the users [17:193].

In summary, while computers built around a single processor have been traditionally used, physical limits are beginning to taper off the performance increases that can be realized. As an alternative, parallel computers may provide additional performance increases for some applications. The type of parallel computer that used be used for an application appears, at this point, to depend greatly on the nature of the problem.

Image Processing Algorithms

In the ideal case, a computation requiring an amount of time, T, to complete on a serial computer would require a time of T/N on a parallel computer with N processors. This ideal speed-up is, in general, not achievable because most computations cannot be divided into N independent computations and thus necessitates the exchange of intermediate results and prevents achieving the ideal speed-up. However, performance gains can be achieved for many algorithms. Because of large amounts of data used in image processing, parallel processing is considered as a possible answer to the performance needs in this field.

According to Hall (10:2), the field of computer image processing today can be divided into five major areas: enhancement, communications, reconstruction, segmentation, and recognition. Image enhancement and restoration address improving the quality of images for human use. Since the measure of success is the subjective opinion of a human

being, exactly what techniques provide the best results must be determined through testing with humans over a period of time. Reconstruction deals with recreating the form of a three-dimensional object from two-dimensional images. One of the more notable applications of reconstruction techniques is computer tomography used in the medical field (24:615).

Image communication deals with the transmission techniques used in the broader topic of digital communications and human factors issues in image enhancement and restoration. Image segmentation is a process that involves breaking an image into meaningful components so that objects, and relations between those objects, can be identified. Image recognition deals with the task of identifying an object that has been previously "learned."

The following sections present some algorithms that have been used in one or more of the five areas of image processing. The intent is to provide an understanding of the algorithms and some of the applications that make use of those algorithms. There are, however, many applications that make use of these techniques that are not discussed here. The algorithms were implemented for this thesis on the Intel iPSC hypercube, a non-shared memory, large grain computer. Some of the algorithms, such as the Fast Fourier Transform require inter-node communications while others do not.

Fast Fourier Transform. The Fourier Transform is an important analytical tool in linear systems analysis, antenna

design, optics, probability theory, random processes, quantum physics, and boundary-value analysis (5:7) as well as image processing. As its name implies it is a transform: a technique involving converting an expression to an equivalent form, often to simplify analysis. In the case Fourier Transform, the two forms involved are the time domain and the frequency domain representations. Transform methods in general are used in areas such as "sound and music analysis, communications systems design, analysis of mechanical vibrations, ocean wave analysis, statistics and many others" (23:16). The logarithm is a simple example of a transform method that can be used to avoid long division by taking the logarithm of the two quantities, subtracting the two, and taking the anti-logarithm of the result (5:2). In image processing the Fourier Transform is used to convert images to their frequency representation for frequency domain analysis. The Fourier Transform of a time domain function is defined as

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \qquad (1)$$

The reverse process, going from the frequency domain to the time domain is tne Inverse Fourier Transform.

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} dt \qquad (2)$$

Similarly, the two dimensional Fourier Transform and its inverse are defined as

$$F(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-j2\pi(ux+vy)} dx dy \qquad (3)$$

and

$$f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u,v) e^{j2\pi(ux+vy)} du dv \qquad (4)$$

where x and y are the spatial domain variables and u and v are the frequency domain variables.

It has become convenient to compute the Discrete Fourier Transform (DFT), the discrete-time version of the Fourier Transform, with the advent of digital computers. Figure 7 illustrates the relation of the Fourier and Discrete Fourier Transforms. Since digital computers can only store and manipulate discrete quantities both the time domain and frequency domain representation must be converted to discrete representations. In Figure 7, step (a) shows a function h(t) on the left and its Fourier Transform H(f) on the right. A sampling function (b) will be used to sample h(t). In the

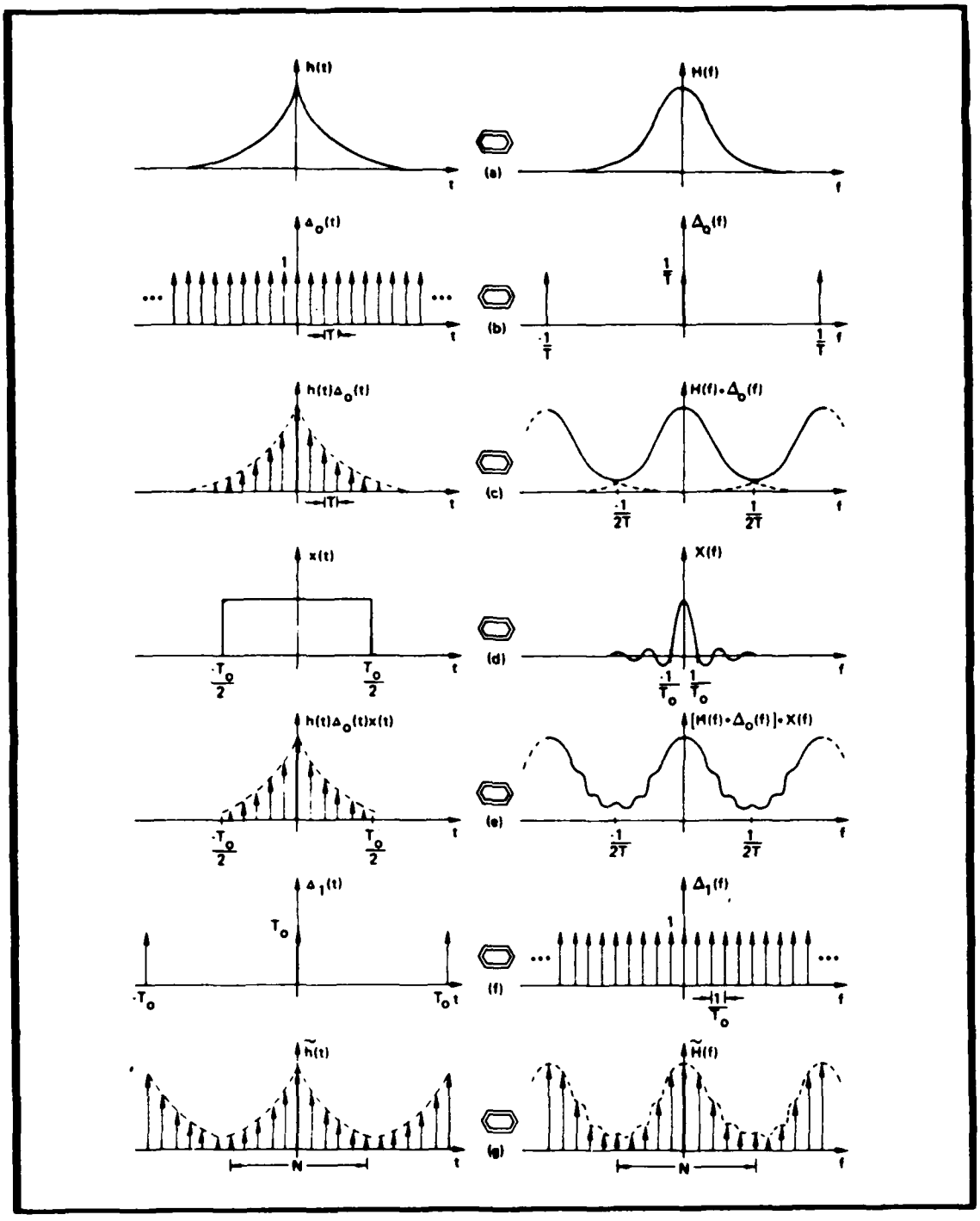


Figure 7. A Graphical Development of the DFT (5:100)

time domain (a) will be multiplied by (b). This corresponds to a correlation in the frequency domain. Next, the time

function is limited in time by multiplying it by (d). At this point the time domain representation of h(t) has been converted to a form suitable for computer representation (e), however, the frequency domain equivalent is still a continuous function. As the final step (f) the frequency domain representation is sampled just as the time domain representation was. The final representation in (g) is a discrete, time-limited, periodic version of the original h(t).

Although the DFT has proven to be a useful tool, the number of computations required for large problems is prohibitive. In 1965, Cooley and Tukey published a paper describing a method that substantially reduced the number of computations required to compute the DFT (6:297). Along with other similar methods proposed later, this method became known as the Fast Fourier Transform. Any of these operations (the Fourier Transform, the DFT, or the FFT) can be applied to a single dimension, such as a voltage level that varies with time, or to multiple dimensions as in image processing where a two-dimensional representation of a scene is considered. The DFT and it inverse are defined as

$$F(u) = \sum_{n=0}^{N-1} f(n) e^{\frac{-ju2\pi n}{N}} \qquad 0 \le u \le N-1 \tag{5}$$

and

$$f(n) = \sum_{u=0}^{N-1} F(u) e^{\frac{ju2\pi n}{N}} \qquad 0 \le n \le N-1 \tag{6}$$

The two dimensional version of the DFT pair is

$$F(u,v) = \frac{1}{N} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} f(n,m) e^{\frac{-j2\pi(un+vm)}{N}} \qquad \begin{matrix} 0 \le u \le N-1 \\ 0 \le v \le N-1 \end{matrix} \tag{7}$$

and

$$f(n,m) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) e^{\frac{j2\pi(un+vm)}{N}} \qquad \begin{matrix} 0 \le n \le N-1 \\ 0 \le m \le N-1 \end{matrix} \tag{8}$$

The Cooley-Tukey method reduces the number of operations from N^2 to $2N\log_2 N$ (6:297) for a one-dimensional Fourier Transform. This reduction is achieved by "decomposing the computation of the discrete Fourier transform of a sequence of length N into successively smaller discrete Fourier transforms" (16:286). This decomposition of the Fourier Transform is generally based on one of two properties of the DFT computation: 1) sines and cosines are used and are symmetric; and 2) periodicity of the complex quantities involved in the computation (16:286). It is the second property that Cooley and Tukey used in their algorithm.

The periodicity of the DFT complex terms can be exploited in either the time domain or the frequency domain (2:14). The first method, decimation in time, splits the input data into even and odd terms. The total transform is

obtained by taking the transform of the two series, multiplying the second by a "twiddle" factor, and taking the sum. This procedure can be applied at successive levels until only two individual transform terms remain. The second method, decimation in frequency, breaks the input data into two halves. Again, this procedure is applied at successively lower levels. "The FFT algorithm derives its efficiency by replacing the computation of one large DFT with that of several smaller DFTs" (15:86). The decimation in time and the decimation in frequency algorithms both require the same number of operations and a reordering of the output data if the natural order is required. An alternative is to preprocess the data such that the output is in its natural order.

In the previous discussion the number of data points, N, was required to be radix-2 or $N=2^t$, where t=1,2,3,... If N is chosen as radix-4, $N=4^t$, further savings in computations can be realized (15:93). This additional savings does come at a cost: N must be chosen from a certain set of values (4, 16, 64, 256 ... for radix-4) and requires supplying zeros for unused data locations, thereby effectively negating any speed-up. However, variations on the FFT exist such that "significant time savings can be obtained as long as N is highly composite; that is, $N = r_1 r_2 \ldots r_m$ where r_1 is an integer" (11:549).

Most existing FFT algorithms have been implemented on computers with one processor, a serial computer. If additional processors are applied to solving the problem then a reduction in the time required to compute the FFT can be achieved. As early as 1968, the possibility of factoring the FFT into independent computations for calculation on a parallel computer was proposed (18:252). This variation of the FFT was design for a hypothetical special-purpose computer capable of vector addition and subtraction, a specific reordering of data, and complex multiplication. As computer technology has advanced and new computer architectures (both parallel and serial) have been implemented, a number of variations on the FFT have been proposed. For example, some of these FFTs require that a matrix be transposed while others do not. Which FFT algorithm is best depends largely on the architecture of the computer used in implementation.

The Fast Fourier Transform (FFT) is a computationally efficient approach to calculating the Discrete Fourier Transform (DFT). Reduction in the number of calculations is achieved by taking advantage of the DFT's periodic properties. The best way to segment the calculation depends on the characteristics of the particular computer to be used. These characteristics include the number of processors and the inter-connection of multi-processor computers.

Convolution. The convolution integral is used in linear systems analysis to determine the response of a linear shift

invariant (LSI) system to an input, given the system's impulse response h(t). That is, a linear system can be described completely by its impulse response. For the input f(t), an LSI system with a impulse response h(t) produces output

$$g(t) = \int_{-\infty}^{\infty} f(\tau)h(t-\tau)d\tau \tag{9}$$

In linear position invariant (LPI) optical systems, the equivalent to the impulse response is the point-spread function h(x,y): the response of the system to an impulse point of light (10:33). Given an input f(x,y) the output of the LPI optical system is

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)\, h(x-\alpha,y-\beta)\, d\alpha d\beta \tag{10}$$

An alternative to computing the convolution integral, or summation in the discrete case, is to use frequency domain techniques. In the frequency domain the convolution integral becomes

$$G(\omega) = F(\omega)H(\omega) \tag{11}$$

Thus, a system's response can be determined by taking the Inverse Fourier Transform of G.

For discrete systems Oppenheim and Schafer present two types of convolution using the DFT as a computation method (16:115). The first is circular convolution. As illustrated in the graphical development of the DFT (Figure 7), the discrete time domain function is periodic with period N. If the convolution summation for two functions is computed using this representation, values from multiple periods will be included in the sum. This result does not correspond to the result that would be achieved for the continuous function convolution. This leads to the need for the second type of convolution, linear convolution. In order to isolate the convolution summation from adjacent periods, each period must be expanded to 2N with the last N values being equal to zero. Thus, when the convolution summation is computed only values from one period will be considered. So, the discrete two dimensional convolution of two MxN arrays is

$$g(x,y) = \sum_{\alpha=0}^{N-1} \sum_{\beta=0}^{M-1} f(\alpha,\beta)\, h(x-\alpha, y-\beta) \qquad \begin{matrix} 0 \le x \le N-1 \\ 0 \le y \le M-1 \end{matrix} \qquad (12)$$

However, if a function X that is to be convolved with another function is limited in its size such that h(x,y) = 0, for x > P and y > Q, the range of summation in Eq. 12 can be reduced to

$$g(x,y) = \sum_{\alpha=0}^{P-1} \sum_{\beta=0}^{Q-1} f(\alpha,\beta) h(x-\alpha, y-\beta) \qquad \begin{matrix} 0 \le x \le N-1 \\ 0 \le y \le M-1 \end{matrix} \quad (13)$$

Therefore, convolving a limited size PxQ function (a "kernel") with the larger NxM function can be reduced by the factor (P*Q)/(N*M), and the summation may be more efficient than using the DFT method. If the functions to be convolved are distributed across the nodes of a parallel processor large numbers of data transfers may be required. The convolution of a PxQ kernel with an MxN function distributed in strips of rows across L nodes will only require nearest neighbor data swaps as long as

$$\frac{M}{L} \ge \frac{(P-1)}{2} \qquad L=1,2,3,... \quad (14)$$

For larger kernels data must be transferred across more boundaries. This results in more memory usage (to store the common data transferred), more inter-node communications, and hence a more complex programming task. Thus, the additional overhead of inter-node communication may further reduce the largest kernel size that can efficiently use the convolution summation as opposed to the DFT method of computing the convolution.

Correlation. The correlation integral is used to measure the degree of similarity between two functions. For

two continuous one-dimensional functions the correlation is defined as

$$g(x) = \int_{-\infty}^{\infty} f_1(\tau) f_2(x+\tau) d\tau \tag{15}$$

For discrete functions the integral becomes the summation

$$g(n) = \sum_{m=0}^{N-1} f_1(m) f_2(n+m) \qquad 0 \le n \le N-1 \tag{16}$$

Expanding to two-dimensional functions the final form is

$$g(x,y) = \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} f_1(n,m) f_2(n+x,m+y) \qquad \begin{matrix} 0 \le x \le N-1 \\ 0 \le y \le N-1 \end{matrix} \tag{17}$$

Note that the correlation differs from the convolution summation (Eq. 12) only in the sign of the integration variables in the second function. Since this is the case, the correlation can be computed in the frequency domain in the same way as the convolution. It can be shown that the axis reversal of the function in the spatial domain is equal to taking its complex conjugate in the frequency domain (10:127). Thus,

$$G(u,v) = F_1(u,v)F_2^*(u,v) \tag{18}$$

So, similar to the convolution the correlation can be obtained by taking the Inverse Fourier Transform of the result.

Image correlation is one of the two basic image matching algorithms, feature matching being the other (22:12).

Histogram. The histogram h(n) is a discrete function that represents the number of pixels at each intensity level in an image. For an image with 8 bits of information there are 2^8 = 256 intensity values. The histogram would have one value to represent the number of pixels at each of these 256 levels. For an NxM image there are N*M pixels so

$$\sum_{n=0}^{255} h(i) = NM \tag{19}$$

Scaling Transform. A simple way to increase the contrast of an image whose histogram does not span the entire range of intensity levels is to scale it. For an image with values in the range (m,M) but, whose histogram can lie within (n,N) the scaling transform is

$$g(x,y) = \{[f(x,y) - m]/(M-m)\}[N-n] + n \tag{20}$$

Logarithmic Transform. The log transform is another useful contrast enhancement technique. Each pixel f(x,y) is replaced with the value g(x,y) = log f(x,y). As show in Figure 8 the values near f_{max} are compressed into a smaller

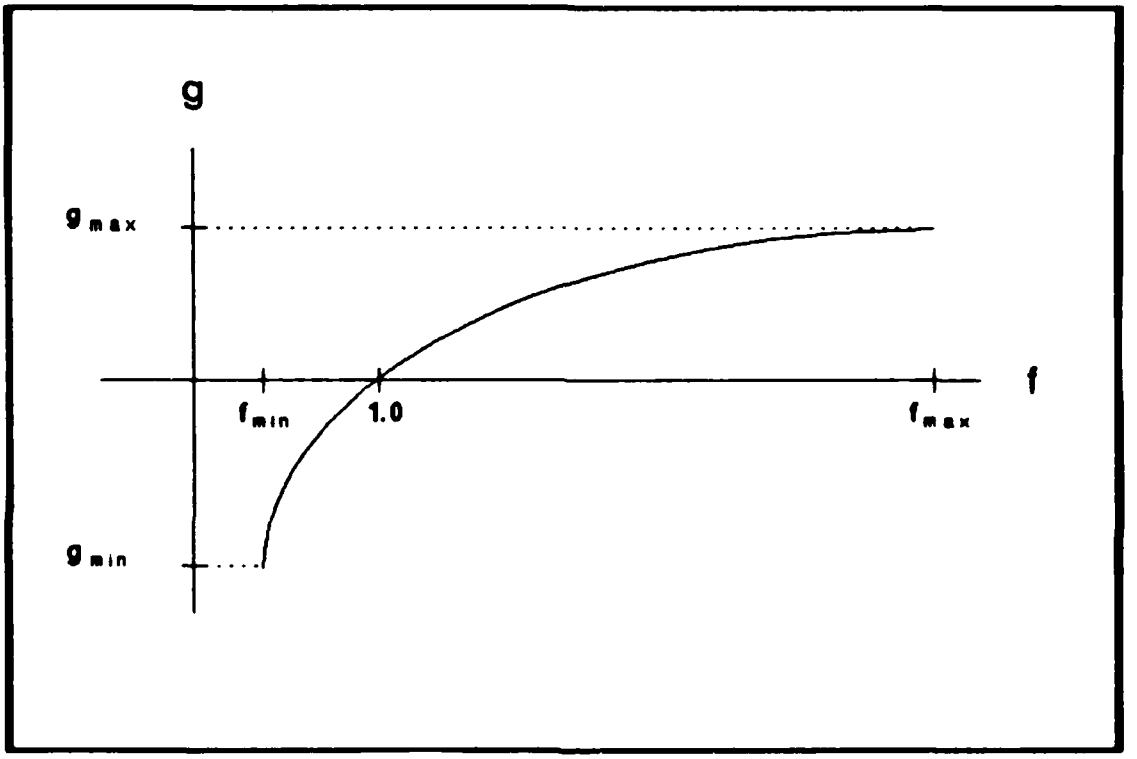


Figure 8. Logarithmic Transform

range while those near f_{min} are expanded into a larger range. Note that varying f_{min} greatly effects the amount of expansion of the smaller values.

Histogram Equalization. Another contrast enhancement technique is histogram equalization which eliminates spikes in the histogram and provides a more uniform use of the range of values. After the histogram h(n) of an image is obtained,

the first step is to compute the "empirical distribution function" (10:171).

$$p(n) = \sum_{i=0}^{n} \frac{h(i)}{MN} \qquad 0 \le n \le I-1 \tag{21}$$

The histogram equalization mapping m(n) is obtained by scaling

$$m(n) = integer\,[p(n)I - .5] \tag{22}$$

Each pixel of the image f(x,y) is mapped to a new intensity value g(x,y) using m(n) as follows.

$$g(x,y) = m(f(x,y)) \qquad \begin{matrix} 0 \le x \le N-1 \\ 0 \le y \le M-1 \end{matrix} \tag{23}$$

Matching Transform. Hall presents a method of combining images through the use of a Hotelling Transform (10:181). If two images are considered, a two dimensional histogram of the co-occurent intensity levels, $h(f_1,f_2)$, can be formed. $h(f_1,f_2)$ is defined as the number of pixels that have a value f_1 in the first image but, have a value f_2 in the second image. This function indicates how closely the two images match. If they are exactly the same then the only non-zero values would lie in a straight line. For images that are not

the same, the Hotelling transform is used to determine the best straight line fit to the distribution.

Interpolation. Sometimes it is useful to enlarge one section of an image. A simple way to do this is by replicating pixels. For example, a 2X magnification requires each pixel be doubled in size in both the horizontal and vertical directions. This method is simple to implement, but the resulting images are grainy and not of much additional use. A better technique is interpolation.

It is well known that a signal x(t) that is band-limited with a maximum frequency f_o can be correctly reconstructed from its sampled values as long as the samples were taken at rate greater than or equal to $2f_o$, know as the Nyquist rate. Therefore, it follows that any intermediate values could also be reconstructed from the samples. For example, from the N samples of x(t) taken at $2f_o$, the 2N samples that would have been obtained had x(t) been sampled at $4f_o$ can be reconstructed. Note that x(t) was confined to a maximum frequency of f_o and no new information is add by the interpolation. Interpolation by a factor L (L = 1,2,3,...) is accomplished by placing (L-1) zeros between each original sample value (up-sampling) and passing the resulting sample stream through a digital low-pass filter. The output of the low pass filter will contain the original values plus interpolated values. For large amounts of data, in the case

of digital images it is beneficial to perform the filtering in the frequency domain.

Summary

Traditionally, computers have used a single processor, von Neumann architecture. However, as physical limitations slow the rate of increase in computational capabilities other solutions are being sought. One of the possible solutions is parallel processing. There are various type of parallel processor architectures but, one of the most versatile is the hypercube. The hypercube has a moderate processor/channel bandwidth ratio in comparison to other topologies and maintains a moderate ratio as more processors are added. The hypercube has several properties that make it a general purpose architecture.

Digital image processing is a field that is beginning to exercise the capabilities of parallel processors. Real-time systems will require an increased computational capability and parallel computers may be the solution.

The image processing field can be broken into five general areas: enhancement, communications, reconstruction, segmentation, and recognition. Frequency domain analysis is an important tool in each of these categories of image processing as well as in many other fields. The development of the Fast Fourier Transform (FFT) has made frequency domain analysis practical by providing a fast digital equivalent to the continuous Fourier Transform. The FFT can be used to

compute convolutions, correlations, and to perform interpolation and frequency domain filtering. Histogram based techniques provide methods to perform contrast enhancement as well as other operations such as segmentation.

III. System Definition

Design Requirements

The overall goal in developing this software package was to provide an environment for digital image processing research. The types of applications that were targeted for implementation under this environment are real-time image processing on multiple image sets. This includes multi-sensor fusion and temporal image processing. In both cases the multiple images considered can be processed simultaneously and then some control action taken or some decision made based on the results. These types of systems include all of the elements of simpler image processing applications and was therefore used as a design requirements goal. Figure 9 shows a block diagram

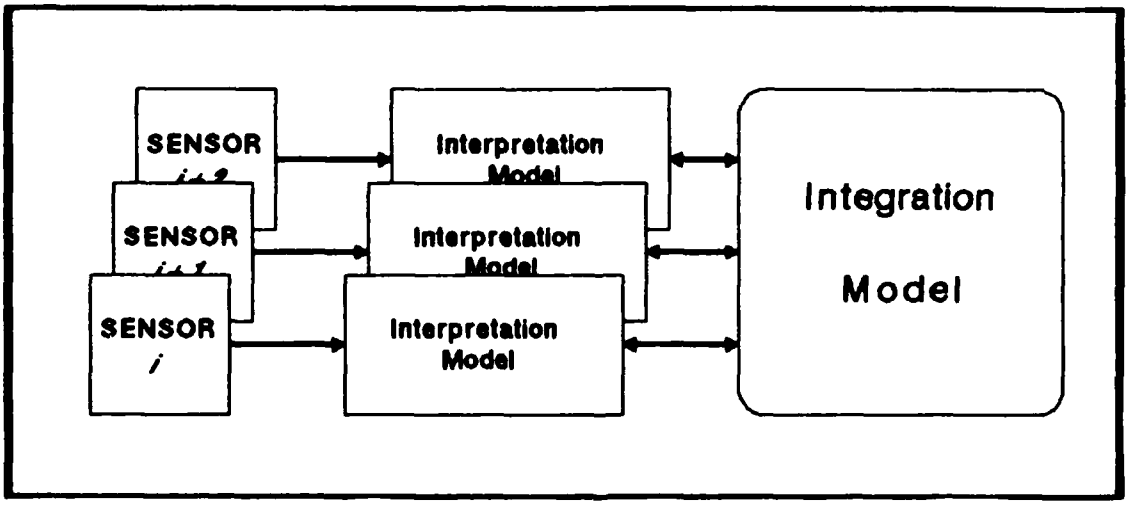


Figure 9. Block Diagram of a Multi-Image System

representing a possible decomposition of the basic elements of such a system. The system depicted uses one or more sensors as its input and may use a single image from each sensor or a set of images separated in time to reach a conclusion. Each image stream is processed based upon an interpretation model specific to the type of sensor. The results from the analysis of the image data is then combined to arrive at a conclusion.

The large amounts of data involved in image processing make implementing real-time systems difficult. First, high bandwidth digital data channels must be used to collect the data for processing and second, vast computational capability must be available to process the data. In designing an environment to develop such applications several desirable characteristics should be noted.

First, the environment should be easy to understand and easy to use. A tool can not be used in an effective way if it is too complex to understand nor will it be used at all if it is more cumbersome to use than present methods. Next, the environment should be robust; it should be able to detect, recover, and indicate errors induced by improper actions or invalid data. The environment should also contain elements that make it generally useful. That is, it should not be designed for a specific sub-set of applications. Next, the environment should be flexible. Digital image processing is a relatively new field and most techniques today are ad hoc.

New methods are constantly being developed so the environment should be flexible enough to be expanded to include these new techniques. On the other hand, it should also be designed such that it is possible to tailor out any unneeded overhead once the application design has been established. Finally, the environment should have a well-defined and consistent interface. This will allow for continued up-grading to its capabilities while maintaining compatibility. This will also simplify efforts to include this environment into a larger structure.

Design Analysis

One way to achieve ease of use and understanding is to break the environment into modular components so that while developing an application, pieces can be added one at a time. The elements of the application can be grouped at conceptual levels allowing the developer to deal with only resonable sized pieces at one time.

Figure 9 is representative of three phases: image acquisition, image evaluation, and an integration of the information derived from the evaluation. As depicted, the data channels for image acquisition from the sensors are independent so multiple parallel input channels should be used to achieve the high bandwidth required to approach real-time processing capability. The information evaluation stage consists of multiple image interpretation models where each will be designed to incorporate the techniques that are

appropriate for that type of sensor. Certain similarities will exist between the different interpretation models since certain fundamental processing techniques will be common. In the last phase, operations will be carried out to relate the pieces of information gathered from the sensor sources. These pieces of information from the image interpretation models may be in a variety of forms. One interpretation model may have an entire image that has been altered as its output while yet another interpretation model may produce statistical information about the image that it processed. Thus, research must determine first, what information can be obtained from imaging systems and second how this information can be integrated. At this highest conceptual level, objects and their relations to one another will most likely be used for symbolic processing. Both the image evaluation phase and the image information integration phase for an application will be composed of various levels of algorithms and operations and, the image processing environment developed should be capable of supporting all these elements for research purposes.

The requirements set forth indicate a hierarchal structure for the environment is needed. Furthermore, each tier should have a limited well-defined set of capabilities that are not replicated in other tiers. The basic elements for an image processing research environment would include image acquisition facilities, mass storage devices for image data and results, ample computational resources, display

devices, and high bandwidth input-output channels to allow development of real-time or near real-time applications. A first step in realizing such an environment is implementing a set of image processing "tools" to serve as the basic image interpretation procedures. However, at this point in the history of image processing there is much yet to be learned and most methods of image interpretation, let alone image information integration, are those that have been tried and have proven minimally useful and are not products of mathematical analysis. Yet, certain low level mathematical operations turn up in many of the methods developed thus far and may, in the end, be a part of the methods that will be shown to be optimal.

The next section presents a software structure that will provide a set of low-level functions that can serve as a starting point for image interpretation and integration research, and form the base of an environment for image processing research. The set of operations presented is not comprehensive, but does permit the implementation of the selected representative algorithms discussed in Chapter 2. The set of operations can be expanded to include those needed for other techniques, such as statistical based methods or conversely, the set can be reduced to form specialized image processing programs. Then these programs of various specialties could be grouped and controlled by a higher level process to form, for example, an image interpretation program

for FLIR images. A group of these image interpretation programs could then in turn be controlled at yet another higher level.

System Definition

A program to do low-level image processing, referred to here as an "image processor program", can be decomposed into three functional areas: a control module to translate calls to the image processor program into a series of actions, an input-output module for retrieval and storage of images and results, and an operations module that contains the set of low-level operations to be used for image processing. There is also a set of data structures that are used to store data from the outside world and are acted on by operations module. The structure of this image processor program is illustrated in Figure 10. This structure was chosen to facilitate additions to the program in the four following areas. First, it is anticipated that additional image storage and image acquisition equipment may be added to the system. Identifying the input-output functions for the image processor program as a separate module allows for addition to or replacement of this module. Next, the operations presented in Chapter 2 represent only a sample of the many being used in image processing and the software will need to be expanded in the future. This may also require additional data arrays. The data arrays selected for this design were based on the algorithms selected for implementation, but other algorithms

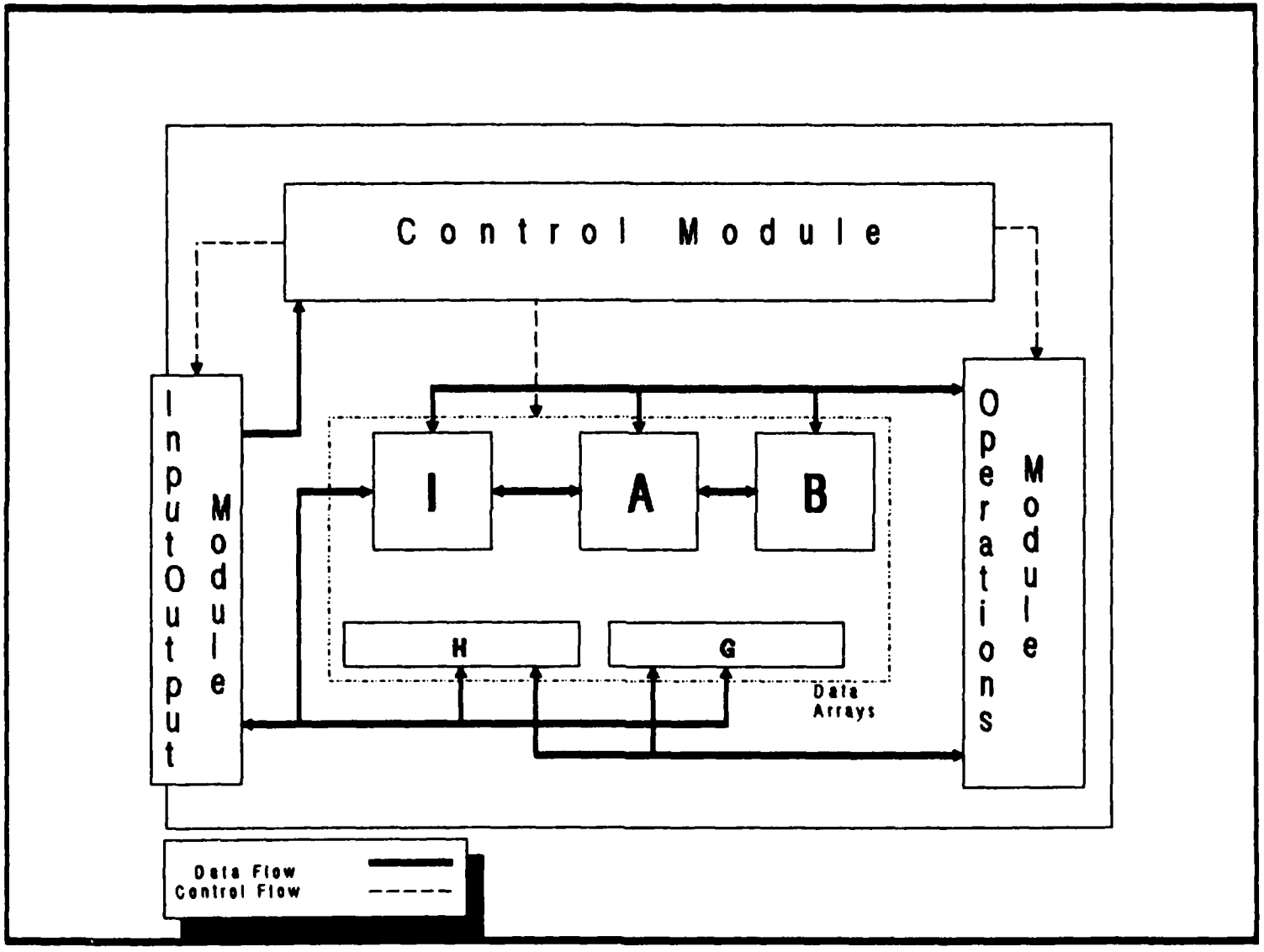


Figure 10. Structure of Image Processor Program

may require more data storage. Finally, all of the above changes would require modification of the control module so it is identified as a separate component.

This structure should allow the program to evolve to suit the needs of a wide variety of applications. The following sections discuss each of the components of the image processor program.

Control Module. The control module receives commands from the input-output module and decodes them. Based on the operation command it will activate one or more of the other two modules to perform a task consisting of one or more steps.

Input-Output Module. As indicated the input-output (I/O) module is managed by the control module. Its function is to pass commands to the control unit and, pass data to and from the data structures of the image processor program.

Operations Module. A set of operations are defined for the data arrays and each effects only certain arrays; each operation is not defined for all arrays in the set. It should be noted that in Figure 10 that no control flows from the operations module to the control module. This is indicative that no branching type operations are possible since the commands are received by the image processor program one at a time.

Data Arrays. The storage of the image processor program is a set of data arrays that was chosen based upon the data requirements of the image processing algorithms and

operations presented in Chapter 2. This set was chosen to permit operations on the original image data, operations on the complex valued equivalent of the image data, or the complex valued data from two images. This set can be expanded as needed within memory constraints. The I data array is NxN array of pixels used to store images where each pixel is in the range 0 to (2^n-1), where n is the fixed number of data bits for the images. The H data array is a array of size $1x2^n$ and whose values range from 0 to N*N. The I and H arrays are used for operations, such as histogram equalization, that are defined over original image data. The A and B data arrays are NxN data arrays of complex values. They are used in operations that require real or complex values for the image such as the Fast Fourier Transform. The G data array is a general purpose complex valued array. Its length is somewhat arbitrary in that it need only be long enough to store parameters needed in operations such as frequency domain filtering. The S array contains information about conditions, such as errors, that have arisen during the execution of the last operation.

Operation Set. Given this structure, an operation set must be defined for the image processor program. As previously stated, once a command is interpreted the control module activates one of the other modules. For inputting, data the control module will activate the I/O unit for reading the data and will activate the memory for storing the

data. For outputting, data the opposite will occur. For data movement within the program, only the sub-components within the data arrays need be activated. Finally, to perform operations on the data, the data is retrieved from the arrays and operated on by the operations module. The operation set is defined by the algorithms used in image processing work.

The following are the set of operations that were chosen for the image processor program to permit the implementation of all algorithms presented in Chapter 2. An objective was to keep the set of commands small for this thesis, but to provide for the expansion of this set in future efforts when more exact image processing requirements can be identified. The operations are structured such that they can be applied to any data array of the appropriate type. Some of the operations are composed partially or entirely of other simpler operations and it is possible to decompose one of these operations into a series of simple operations and achieve the same result by issuing that series of commands, but it is beneficial to have the more complex operations if they are often used to decrease command traffic.

I/O Operations. The input-output operations required for the image processor program include getting the next command and inputting and outputting the arrays. These operations are given below.

Get Command - receive the next operation command.

Input I array - input an image data array.

Output I array - output the image data array.

Input H array - input an H data array.

Output H array - output the H data array.

Input G_N array - input complex valued array of length N.

Output G_N array - output the first N elements of G.

Data Moves and Conversions. As indicated above, there are primarily two data types needed for the image processing operations: integer intensity values from the original image and complex values used for FFTs and other similar operations. Operations to convert between the two data types are needed as well as copy operations between the two complex valued arrays A and B. A summary of these operations is presented below.

I <- A - convert and copy the A array into the I array.

A <- I - convert and copy the I array into the A array.

A <- B - copy the B array into the A array.

B <- A - copy the A array into the B array.

A <-> B - exchange the A and B arrays.

Single Image Operations. There are a number of manipulations that are commonly used in manipulating a single image. Some are defined for the intensity values of the original image and others for non-integer and complex quantities. The set of single array operations chosen are

A <- 0 - zero the A array.

Re[A] <- |A| - take the magnitude of A and store the result in the real part of the A array.

Re[A] <- arg A - take the phase of the A array and store it in the real part of the A array.

A <- A^* - take the complex conjugate of the A array.

A <- transpose { A } - do a matrix transpose on the array A.

A <- log { A } - do a logarithmic transformation on A.

A <- fft { A } - take the two-dimensional Fast Fourier Transform of A.

A <- fft^{-1} { A } - take the inverse 2D-FFT of A.

A <- A + G_1 - add the first element of the G array (a complex constant) to the A array.

A <- A · G_1 - multiply the A array by a complex constant.

A <- scale { Re[A] } - perform a linear scaling operation on A such the the real component of A lies in a specified range.

A <- filter { A } - bandpass filter the (frequency domain) A array.

G <- range { A } - store in G the range of values in A.

H <- histogram { I } - calculate the histogram of the I array and store it in the H array

I <- map { I, H } - remap the intensity values of I to the values in H.

Two Image Operations. Some operations involve data from two images. These include arithmetic operations such as addition, or multiplications while other two image operations are more complex, such as the two dimensional

convolution of two images. Note that A and B are complex valued and all arithmetic operations indicated are complex.

A <- A + B - add the B array point by point to A.

A <- A - B - subtract the B array point by point from the A array.

A <- A · B - multiply the A array and B array point by point and store the result in A.

A <- A / B - divide the A array by the B array point by point with all values in B being non-zero.

A <- matching transform { A, B } - performing the matching transform on A and B.

A <- correlation { A, B } - perform the two dimensional correlation of A and B.

A <- convolution { A, B } - perform the two dimensional discrete convolution of A and B.

A <- convolution { A, G_{MxM} } - perform the convolution of A and a MxM kernel stored in G.

Summary

This chapter presents a description of the type of environment that would be useful for image processing research. The image processing applications of particular interest, real-time multiple image processing, require a flexible and powerful set of capabilities in both hardware and software. A hierarchical arrangement of image processing capabilities, ranging from low-level sets of image processor programs to higher level symbolic processing control structures, is proposed as a possible way to meet these requirements. The set of low-level functions needed to implement the image processing algorithms discussed in

Chapter 2 were outlined. This set of image processing operations can be expanded to include other operations as needed.

IV. Detailed Design

This chapter presents the detailed design of the system presented in Chapter 3. In the first section an overview of the iPSC computer is presented. In the next section the image processing software is broken into its logical modules and an overview of how the modules were implemented on the iPSC is given. Next, the method of test for each of system modules is presented with the actual test results being presented in Chapter 5. In the last section the set of functions that were provided to give the user access to the image processing software running on the iPSC is outlined.

iPSC Hypercube

The following description of the Intel iPSC was taken from iPSC system documentation and is provided to give the reader an understanding of the hardware and software facilities of that system. Further information is available in the iPSC System Overview and the iPSC Programmer's Reference Guide.

The iPSC is a family of expandable parallel processing computers that have the MIMD architecture discussed in Chapter 2. Each system is composed of two major components: the "cube manager" and the "cube". The cube manager is an Intel System 286/310 running a Unix-based multi-user, multi-tasking operating system. The 310 has an 80286

microprocessor with the 80287 math co-processor and 2 megabytes (Mbytes) of memory. Mass storage includes a 140 Mbyte Winchester disk, 360 kilobyte (Kbyte) MS-DOS compatible floppy disk, and a 45 Mbyte cartridge tape drive. Multibus boards, Ethernet connections, up to 9 terminals, and up to a total of 5 Mbytes of main are options supported for the 310. Software support for program development includes Fortran, C, LISP, and assembler languages, cube control and diagnostic utilities, and debugging utilities. The cube manager is connected to the cube via a 10 Mbit/sec Ethernet data channel. The cube is an MIMD computational unit which is a collection of non-shared memory processors connected in a hypercube topology (see Chapter 2). Each cube actually consists of one, two, or four separate physical enclosures where each contains up to 32 processors (nodes). Thus, models with 32, 64, and 128 nodes are available. Each node has a 80286 microprocessor, a 80287 math co-processor, and .5 Mbytes of memory. A multi-tasking node operating system supervises loading and terminating of application programs and provides the inter-node communications facilities. Two node options are available with the iPSC. The first is a memory expansion that increases the memory of each node to 4.5 Mbytes. The second is a vector processing expansion that performs vector operations on up to 16K element vectors and provides an additional 1 Mbyte of memory to the node. The memory option and the vector processing option can not be used in conjunction. Nodes are connected in a hypercube

configuration using 10 Mbit/sec Ethernet data channels between nearest neighbor nodes. Communications between the nodes and between the host and the nodes is accomplished with the system's set of synchronous and asynchronous message passing functions.

Application programs typically consist of two programs: a host program and a node program. The host program is coded, compiled, and run on the 310 using message passing to communicate with node programs. The node program is coded and compiled on the 310 and then loaded into the nodes using system commands. While typical applications use the same node program running in all nodes, it is also common to load different node programs into the various nodes of the cube.

The iPSC system used in this thesis is a 32 node model with the vector processing expansion and an Ethernet connection from the cube manager to a network of other computers. A node program to perform the functions described for the image processor program was written in the Fortran, as opposed to C because at this time the vector processing option is only supported in FORTRAN. A set of C functions were written for the host computer to interface with the node program.

System Implementation

As described in the previous chapter, the image processor program is broken into four basic components: 1)

the control module, 2) the input-output module, 3) operations module, and 4) the data arrays. The first three components form the three functional blocks of the design. The data arrays are not global data structures but, rather are data entities within the image processor program that are managed by the control module. The image processor program is an asynchronous non-deterministic program with local memory. That is, once the image processor program is started it will continue to run, processing commands upon receipt while maintaining its data array's contents. This is in contrast to the majority of software systems running on uni-processor computers where programs (typically called subroutines, procedures, functions, etc.) start running only when they are called and do not retain the values of internal variables from previous calls. One exception is the Ada language which supports independent, concurrent programs called "tasks" (4:63). Since Ada is generally used on uni-processor machines, these independent program structures are run by time-sharing the single processor.

In the image processor program code developed for this thesis, images are divided into horizontal strips of equal size and distributed across the nodes, giving each slice two adjacent slices. Any operations that require data outside a node's slice must do a data swap with the node containing the required data. Thus, this division is advantageous in two respects. First, it is easier for the programmer to

implement the code by reducing the number of boundaries that data must be transferred across. Second, the number of messages in the cube is also reduced, thereby avoiding data channel congestion from larger numbers of messages.

For this thesis, 16 nodes were used because the FFT routine written by Intel was designed to run on a cube of this size (a recommendation in Chapter 6 addresses rewriting this code). However, all other routines were written to be used on cubes with two or more nodes. As discussed in the quote from Palmer in Chapter 2, the hypercube architecture is "recursive". That is, a hypercube of dimension N is composed of 2 hypercubes of dimension N-1. This makes it easy to write code that can be run on hypercubes of various sizes.

Control Module. The two basic functions of the control module are to receive and decode commands, and call the other modules of the image processor program to perform a series of actions based upon the commands. Once the image processor program is started it will repeat this two part process indefinitely. This translates into the following program structure.

```
loop forever
   get_command
   case (command)
      command_1 :   . . .

      command_2 :   . . .

      command_3 :   . . .
```

```
                    .
                    .
                    .
          command_N :    . . .

        end case
    end loop
```

For each of the commands, some sequence of calls are issued to the input-output module and the operations module. The control module was first implemented and later, operations and I/O functions were added one at a time. The commands consist of integer values that are passed from the software on the host to each of the nodes. Each node then executes the appropriate steps to carry on that operation. A status flag was included as a part of the image processor program to indicate the success of the most recent operation and can be read via a "status" command provided with the host routines. Test cases for this module need only consist of sending valid commands and representative examples of invalid commands and checking for the appropriate response.

Input-Output Module. The input-output module is the link to the host program for the image processor program. Two classes of data are passed through the input-output module: commands, which are received and passed to the the control module and secondly, data arrays, which can be input to and output from the data structures. Each node receives only a portion of the image data while the smaller arrays are replicated in all nodes. The input-output module was

implemented as a set of input and output pair of routines for each type of data array. The routines were written to apply for any instantiation of that data array type so that additional arrays can be added to the program if desired.

Verification of the correct operation of the input-output functions can be accomplished by inputting known values and checking the output arrays for consistency with the known values.

Operations Module. The operations module is the major functional part of the image processor program and consists of the set of elementary operations that are necessary to implement the image processing algorithms. The following paragraphs present the hypercube implementation of each of the operations listed in Chapter 3.

A large portion of the operations implemented required no data exchanges between slices. These include all operations that involved point by point operations such as copying one array to another, point by point arithmetic operations between arrays, and arithmetic operations on arrays involving a constant. These operations are the simplest to implement because each node merely applies the particular operation to its portion of the array as if that where the entire array. The more interesting and difficult operations are presented in the following subsections. Verification of the correctness of the results of these operations is complicated by the large sizes of the image data array, so the operations were validated by using test

cases for which the correct results were known. The details of these tests are presented in Chapter 5.

Matrix Transpose. The matrix transpose, the most complex operation presented, was written by Intel programmers. The data arrays are distributed across multiple nodes and hence a matrix transpose requires each node to send part of its portion of the matrix to each other node. This constitutes a complete interchange of data and is the most communications intensive type of data exchange in parallel processors of any type. The matrix transpose routine provided by Intel takes advantage of the message routing algorithm used in the iPSC and a "synchronization" routine to maximize communication channel usage while minimizing data channel contention. This function was written for a cube with exactly 16 nodes and is not easily adaptable to other cube sizes. This restriction stems from measures required to overcome deficiencies in the message traffic throughput capabilities at each node in the iPSC system. However, these deficiencies are expected to be alleviated in the next version of the hardware and consequentially the transpose function can be simplified at that time.

Two-Dimensional Fast Fourier Transform. The 2D-FFT was also implemented by Intel. It uses a one-dimensional FFT routine built into the vector processing board and the matrix transpose. Using the "row-column method" (16:320), the 2D-

FFT is computed by performing one-dimensional FFT on each row of the matrix, transposing the matrix, and again performing the FFT on each row. The result is the transpose of the 2D-FFT so the matrix must be transposed a second time to get the proper form.

Convolution and Correlation. As discussed in Chapter 2, there are two ways to perform the convolution or correlation between two images. In the first the summation is performed in the spatial domain. As will be shown in Chapter 5, for cases where an image is convolved or correlated with a smaller sub-image this method is satisfactory but, for two full sized images it is faster to perform the operation in the frequency domain. To convolve two images in the frequency domain, the 2D-FFT of both images is computed, the two FFTs multiplied point by point, and the inverse 2D-FFT of the product is computed to yield the result. The correlation is similarly computed but, the complex conjugate of one of the images is taken before the two FFTs are multiplied. Both spatial domain and frequency domain convolutions were implemented while only a frequency domain correlation was implemented. The frequency domain operations were implemented using the functions already discussed. The spatial domain convolution requires the sum of the product of the image and sub-image for each point in the larger image. Thus, for points within a certain distance of a slice boundaries a data exchange is required. For example, when using a 7x7 convolution kernel, calculating the

top edge values for a horizontal image slice requires a horizontal strip 3 pixels tall from the image slice above. This is accomplished by exchanging the required boundary pixels across every boundary and storing them in a buffer to be accessed during tne convolution summation when needed.

Histogram and Histogram Equalization. The histogram functions implemented make good use of the parallel architecture. In forming the histogram, each node forms a histogram of its own slice and tne total histogram is formed at the host by summing the partial results. The histogram can, if desired, then be equalized in the host machine. The new histogram is then sent as the input to the mapping function which distributes this histogram to all nodes. Each node then maps its local pixels to the new values indicated by this histogram.

Matching Transform. The first step in performing the matching transform is to find the mean of both images. This is accomplished by having each node performing a sum of its own pixels, totaling the partial sums in node 0, and then dividing by the total number of pixels. Node 0 then distributes the mean to all other nodes so each can form its part of the covariance matrix summation. The contributions from all nodes are again summed in node 0. Node 0 alone proceeds to find the eigenvalues of this 2x2 covariance matrix. Using the largest of the two eigenvalues, node 0 finds the eigenvector. The two elements of the eigenvector

are then distributed to all other nodes. Each node uses these values as weights for adding the two images together. Thus, the pixel by pixel operations are distributed across all nodes, while summation of partial results and simple calculations are left to a single node.

Frequency Domain Filtering. The filtering routine was taken from code written by Intel programmers. First, a one-dimensional bandpass filter, of length N for an NxN image, with a $\sin^2$ roll-off is generated based on user selected parameters specifying the pass-band region and a roll-off region to either side of the bandpass region. This filter is then applied to first the rows and then to the columns of the frequency domain representation of the image. No data is exchanged between nodes because each node generates the entire one-dimensional filter for its own use. Thus, given that a one-dimensional filter F(n) has been generated, a pixel at an arbitrary position (x,y) in the image is first multiplied by F(x) and then by F(y) effecting the same results had a two-dimensional filter been generated. Since each node can determine where in the image its slice comes from, the proper index values x and y can be calculated in terms of the whole image, not just the local position.

Host Program Interface

A set of C functions were written to provide a simple to use interface for writing applications that would use the image processing procedures presented above. The procedures

use a set of three types of message passing routines. The first is a function that passes commands to the node programs. The second type of routine sends and receives the image data. The third type deals with the integer array used for histogram mapping and the fourth transmits the complex array used for filtering parameters, constants, and other such quantities. Using these sets of routines, functions to perform the FFT, convolution, and the other image processing operations were written. The image processing routines on the host machine can be viewed as a bridge between the application and the image processing routines in the cube. This interface removes the burden of dealing with the multi-processor computer and presents a "single system image" (1:31) to the user.

V. Software Analysis

This chapter describes the steps taken to insure the image processing routines implemented are correct and presents some timing measurements of run times for the FFT for various computers. In the next section, some information about the difficulties encountered while programming the iPSC is given and finally, an assessment of the package as a whole is given.

Correctness Tests

Verifying the correctness of software is a difficult task and has been the subject of much literature (20:318). In the case of image processing software, the difficulty of this task is compounded by the large size of the data files. For example, it is practically impossible to analytically compute the 2D-FFT of a 256x256 image. Thus, in many cases, the correctness of a program can only be assessed by comparing the output with results obtained by others.

Two basic methods were used in testing the routines. First, the images were visually inspected to determine if the expected result was obtained. While this method is not precise, it is a good indicator. The second method used was comparing the maximum and minimum values of the resulting image with the maximum and minimum values expected. The combination of these two techniques proved an effective way

to detect errors in the routines. The following sections outline the procedures used to verify the correctness of each of the particular routines.

Point-by-Point Functions. Many of the operations implemented required nodes to operate only on its own data and did not require inter-node communications. These functions were tested by inputting images with a known range of values, performing the operation, and verifying the range of the resulting image was as expected. This method was augmented with inspection of the resulting images.

Fast Fourier Transform. The FFT routine written by Intel and incorporated into the image processing software was tested first by transforming then inverse transforming several images to verify the original image was preserved. As a further test, an 3x8 square with a constant amplitude across its surface was transformed and the log transformed magnitude of its frequency representation was compared with the results presented by Pratt (19:477). The result was a two dimensional (sin x)/x function which follows from the one-dimensional transform of a rectangular function (29:493). As larger squares were input, the spacing of the zero crossings in the frequency domain decreased as expected. The correctness of the inverse FFT follows from the correctness of the FFT since images were correctly recovered from the frequency domain.

Correlation. It is the property of the correlation function that if a "template" is placed in the upper left-

hand corner of an image and is correlated with a second image, the "scene", peak values will occur in the resulting image at locations corresponding to the locations where the pattern occurred in the scene. Thus, the location of these "correlation peaks" are an index into the scene to occurrences of the pattern. The correlation function was tested by using various shaped patterns and correlating these patterns with scenes that contained copies of the pattern.

Convolution. As mentioned in Chapter 2, the convolution and the correlation differ only by taking the complex conjugate of one image in the case of correlation. Thus, since the same code was used for both the convolution and the correlation with only a change it in signs of two terms in the multiple (to effect complex conjugatation) and the correctness of the convolution follows from the correctness of the correlation. But as a further check, squares of various sizes were convolved to insure the convolution worked as expected.

Convolution Kernels. The 3x3, 5x5, and 7x7 convolution functions where checked by operating on images with various kernels and inspecting the resulting image and checking the range of values to verify the results. The first kernel used was an impulse, that is, the kernel contained a single non-zero value equal to 1. This function passed the image with no changes except for offsets corresponding to the offset of the 1 in the kernel. Thus, if the single non-zero term is in

the center of the kernel no offset resulted. Next, the non-zero term in the impulse was changed to values not equal to 1 and the range of resulting image was verified. As a final test, various know operators (13:9.1) such as edge enhancers and high pass filters were used on images and the results verified by inspection.

Band-pass Filter. The operation of the bandpass filter was verified by two means. First, an image with constant intensity values through-out was used as the input to the filter operation and the result was inspected to verify that the regions of the image left at the constant value corresponded to the location of the pass-band frequencies. Next, the filter was used on the frequency domain representation of an image and the spatial domain result inspected for the correct alterations. Images were inspected for "ringing" around edges in the cases of sharp cut-offs for the filter.

Histogram. It is the property of the histogram that, for an NxN image, the sum of the histogram values will equal N^2. This property was verified with various images as input. One of these images was a KxK square of uniform intensity Q. The resulting histogram indicated K^2 pixels of value Q and $(N^2-K^2)$of value 0.

Histogram Equalization. The histogram equalization function was tested by comparing a portion of the histogram of an equalized image with the results of a hand calculation of the equalization process. As a further test, an image

with a low intensity region containing details was histogram equalized to verify the region's intensity was increased as expected.

Matching Transform. The matching transform function was tested by inputting electro-optical and FLIR data from a scene and visually comparing these with the results obtained by Hall (10:182). As a further test, two copies of an image were used as input to the matching transform. As expected, the two copies were equally weighted and the resulting image was equal to the original.

Performance

The following time measurements were taken to give a general indication of the relative speed of the image processing package. The measurements are not necessarily the best possible on each machine, but do provide a rough comparison between the options that are typically available to a programmer during the development of an image processing application. Any attempt to represent the performance of a multi-processor computer in terms that can be compared with the performance of a uni-processor computer or even another multi-processor computer would require many assumptions and caveats and thus, would prove useless as a basis of conclusions. Thus, the times below are presented as the times required to do particular computations with means that were readily available and not as general performance measures.

In all measurements below for the 2D-FFT the "row-column method" (16:320) was used.

170.6 sec - C program on Sun 3 workstation.

38.1 sec - C program on Vax 11/780.

20.0 sec - C program on Sun 3 workstation with MC68881 math coprocessor.

7.2 sec - Fortran program on iPSC hypercube with 16 nodes.

.6 sec - Fortran program on iPSC hypercube with 16 nodes and vector processor boards.

Time measurements on the iPSC were taken for a 16 node configuration of the Fortran node program. After the FFT operation command, a status request was sent: the times on the iPSC were measured from the time the operation request was made to the time the status reply was received. The previously measured overhead time of the status message was then subtracted to determine the net time for the operation. It is important to note these times do not include times to load and retrieve images, which total to about 2 seconds. Times for all other machines are cpu time for a C program using the same FFT algorithm, obtained by using the times() function.

The 8.5:1 ratio between the Sun workstation times with and without the MC68881 indicate the large number of mathematical calculations in the 2D-FFT. The 12.67:1 ratio between iPSC times with and without the vector processor boards reflects a gain in the speed of mathematical

calculations, but also a gain due to the additional parallelism of the vector processors at each node.

The following time measurements were taken for the convolution of various sized kernels with a 256x256 image. The ITEX hardware (13) is a dedicated add-on board for a micro-VAX II for image processing. The times measured for the iPSC were for a 16 node configuration using the vector processing capabilities.

For a 3x3 convolution kernel the iPSC was faster by a 3:1 ratio.

10.8 sec - ITEX.
3.6 sec - iPSC vector with 16 nodes.

For a 5x5 convolution kernel the ratio was reduced to 2.5:1 ratio.

20.8 sec - ITEX.
8.3 sec - iPSC vector with 16 nodes.

Finally, a 7x7 convolution kernel yielded the following times with a 2.38:1 ratio.

35.5 sec - ITEX.
14.9 sec - iPSC vector with 16 nodes.

This gradual closing of the margin between the iPSC and the ITEX can probably be attributed to either increased

efficiency on the part of the ITEX for larger data set sizes or inefficiencies in the data exchange process for the iPSC.

System Programmability

The programming task on a parallel computer proved much more difficult than programming on serial computers. Much of this difficulty was experienced in data communications, both host to nodes and node to node. The system's message passing routines are of a very elementary form. Parameters to specify a message include, the node's own communications channel number, the "type" of the message, the buffer that is to be sent, the length of the buffer, the destination node, and the process number of the destination node. Since all messages in the system's message passing routines are in terms of bytes, not in terms of the number of data elements, the programmer is required to determine the number of bytes in arrays of various types and this often leads to errors caused by misinterpretation of the buffer contents. For scalable routines, the destination node is often determined by a calculation and this also increased the likelihood of errors. Often the data type of variables was misdeclared and messages were erroneous or lost. All of these factors can be traced back to the programmer's error, but the number of parameters to consider and the difficulty of debugging multiple programs, including the host program, resulted in a substantially more difficult programming task. Booch makes reference to the Hrair limit.

> ... there exists a fundamental limitation in our ability to manage a number of different objects or concepts at one time. In 1954, the psychologist George Miller concluded that the limit to the number of entities humans can process at one time is roughly seven, plus or minus two. ...our ability to manage the complexity of many entities falls off sharply after this number [4:31].

Thus, while the code for the image processing routines was broken into distinct modules, the complexities of programming a parallel computer, with the present system tools, still proved to be a strain to manage. Programming errors, in many cases, took days to find even when the problem was an elementary error.

Image Processing Routines

The previous discussion points up the exact goal that was be addressed when the software structure was designed: reduce the complexity of using a parallel computer while maintaining as much of its versatility as possible.

As described in Chapter 3, the image processing functions were organized around a small number of data arrays with a set of operations that are allowed for these arrays. These "image processors" can then be grouped into a higher level structure, while keeping the total number of entities being manipulated below the Hrair limit.

VI. Conclusions and Recommendations

Conclusions

During this thesis, a software structure that simplifies the use of a parallel computer for image processing was developed. Furthermore, the capabilities of this software can be readily expanded to meet the future needs of image processing researchers. This thesis has demonstrated that parallel processors can be used to perform many image processing algorithms and provide a unique environment for image processing research. This is significant because present sequential processors may soon reach physical limits and other techniques, such as optical processing, have not produced usable techniques for faster sequential processing. Therefore, parallel processing is the best alternative to meet the increasing computational requirements of digital image processing and that of other fields as well.

During this thesis, some of the algorithms implemented have proved excellent candidates for taking advantage of a multi-processor environment because they were perfectly divisible. However, this effort used a relatively small number of nodes. For an image size of 256 by 256 each of the 16 nodes used had only 16 scan lines of the image. If the number of nodes is increased arbitrarily the image becomes so fragmented over the nodes that internode communications for those algorithms that are not perfectly parallel becomes a

prohibitively large proportion of the time required to complete the computation. Therefore, while parallel processing can increase the processing rate of image processing applications it does have factors that govern the exact implementation chosen.

Finally, an observation about programming a parallel computer; while programming a multi-processor computer proved more difficult than programming a uni-processor computer, it is possible given the number of entities being manipulated is keep to a manageable level. Thus, modularity of the code and a hierarchal structure for the code is imperative.

Recommendations for Further Work

Parallel processing is relatively new area and there are many applications that can make use of it. In the area of digital image processing new techniques are being developed each year and many of them rely on algorithms presented in this thesis or similar ones. Future pursuit of the work presented in this thesis can contribute to increasing the speed of these algorithms and could aid in providing a real-time capability for these techniques. Several areas of improvement or additional capability are outlined in the following sections for future work. The first three outlined are contingent on Intel up-grades to the iPSC hardware and software.

Matrix Transpose. In the FFT routine a set of data arrays are used to determine when a node should transmit its part of the image to other nodes. This is effective because the iPSC uses a fixed routing algorithm. This method was required to overcome shortcomings in the iPSC's communications bandwidth. When the new 80386 based node hardware becomes available it will help to ease the original problem encountered but, it will not make the matrix transpose simple. For example, the algorithm

```
for i = 1 to number_of_nodes
  if i not equal my_node_number then
     send_to_node( i , array_part(i) )
     get_a_message( array_part(j) )
  endif
next i
```

would result in all nodes, except node one, sending their first message to node one. This will result in a large congestion in the communications paths surrounding node one. This matrix transpose is in the larger class of problems known as a complete data exchange, that is, each node sends information to every other node. This communications task is the most demanding on any parallel architecture. The data arrays used in the FFT were constructed by Intel programmers for a dimension-4 cube only. While it would be possible to construct arrays for other required cube sizes, a general solution would prove more satisfactory. It should be

possible to construct an algorithm that calculates where the next message should go in a way that minimizes congestion in any one area. Three quantities would be needed in the algorithm

1. The cube size.
2. The originate node's number.
3. A counter to keep track of the process that would take 2^d steps, where d is the cube dimension.

Image Rotation. Rotating an image in a sequential processor is a computational-intensive and memory access intensive operation. The equation for calculating the new position (x',y') of a pixel (x,y) in the image rotated by theta (10:123) is

$$x' = x \cos \text{theta} + y \sin \text{theta}$$

$$y' = -x \sin \text{theta} + y \sin \text{theta}$$

In the frequency domain this becomes

$$u' = u \cos \text{theta} + v \sin \text{theta}$$

$$v' = -u \sin \text{theta} + v \cos \text{theta}$$

This task is not easily implemented in a parallel processor, such as the iPSC, since data must be exchanged at least between adjacent segments of the image or at worst a complete exchange is required.

Multiple Task Division. The present implementation of the image processing environment uses each of the 16 nodes to run multiple processes. This causes a contention for the vector co-processor. When the operating system on the iPSC is up-graded to allow requests for subcubes the multiple processes can each be allocated its own subcube and thereby remove the contention problem associated with the vector co-processor.

Improved Fast Fourier Transform. The FFT-related subroutines used in the node program written by Intel are not scalable because they use a set of data arrays specifically set up for a dimension-4 hypercube to determine which nodes are to exchange data during the matrix transpose. This process is synchronized by using a routine at the beginning of the transpose and passing a token message to signal when node sets can exchange data. These measures where necessary because of the limited message traffic capacity of the present hardware. The next planned release of node board hardware will include a set of node communications co-processors. This will eliminate the need for the extraordinary measures required used for this version of the hardware. The FFT routines should be modified to eliminate its dependency on cube dimension.

Dynamic Cube-size Scaling. The software written for this thesis was written for a fixed cube size due to the FFT subroutines peculiarities discussed above. For future applications it would prove useful to have the node programs

modified so that the routines could dynamically scale all operations based on the size of the cube they were loaded in. Once the FFT routine has been modified as outlined in the previous section, the other routines can be modified to run for an arbitrary cube size.

Dynamic Image-size Scaling. For this thesis a fixed image size of 256 by 256 was used because this was the size of the available data base. While this image size can be changed by simple modification and recompilation of the code, the capability to choose the size of the image could be an excellent enhancement to the image processing environment. A fixed image size was chosen for this thesis to simplify the code development and was planned as the next enhancement to the software.

Image Processor Communications. The slowest operations implemented in the image processing environment are the loading and retrieving of images. A useful operation to provide would be the capability to move information directly from one image processor to another. This would allow data to be consolidated in one image processor when comparisons or a fusion of the data was necessary with having to retrieve and store that data from one image processor and transmit the data to the second image processor. This can be accomplished by adding a two part command to the command set already implemented. First, a message would be sent to the origin image processor program, indicating where its image should

be sent. Second, a simple "get image" command would be issued to the destination image processor program, but no image be sent from the host. This second part requires no additional code for the node program since the present get image command does not care where an image arrives from.

Summary

Parallel processing is a relatively new computer technology that holds promise for solving the problem of ever increasing need for greater computational capability in image processing systems. This thesis has taken a small step in the direction of developing this potential into a solution by providing a tool that others may use in their research.

Appendix: User's Manual for the Image Processing in Parallel Environment

Introduction

This manual outlines the capabilities and use of the Image Processing in Parallel (IPP) environment. The IPP environment is a collection of image processing routines that make use of an Intel iPSC hypercube to reduce the computation time required for these operations and to provide a multi-task environment for image processing. No detailed knowledge of the hypercube or parallel processing is required to make use of the routines. The environment can be used from any Sun 3 workstation running the Suntools environment that has an Ethernet connection to an Intel iPSC Vector cube. This manual focuses on the use of this environment as installed on Sun 3 workstations at AFIT in the Department of Electrical Engineering. At present all of these workstations do have the required Ethernet access to the department's iPSC Vector cube.

Overview

The IPP environment makes available to the programmer a set of image processing procedures written in the C programming language. The environment is presently set up to manipulate 256 by 256 images with 8 bits of resolution for each pixel. The environment allows an image processing

application to start an "image processor program", request a task be performed by the program and proceed in the application while the task is being completed. Later, when the data is needed the results can be retrieved. A "status" mechanism is included to enable the application to query if the results are available. By starting multiple image processor programs the application can specify operations on different data sets and proceed with other tasks while the operations are performed. The application will only be interrupted from proceeding when data from an image processor program is needed, but the operation has not been completed. This method of asynchronous operation allows the user application to act in a supervisor role while IPP image processor programs provide an image processing capability running independently of the application.

While the library provides a powerful mechanism to do image processing, a set-up process to access the hypercube is required prior to using the routines. This process is outlined in a step by step fashion later in the manual and should not prove difficult even for those with little experience using the Unix operating system on the Sun workstation.

Software Installation

An IBM format disk containing source files, executable files, and installation instructions has been provided with this thesis. Have the files on the disk installed on the

hypercube by the system's administrator and type

```
more README.IPP
```

As each screen has been read, use the space bar for the next page or the return key for the next line. This is a text file explaining where the files on the disk should be installed.

Requirements for Using the IPP Library

There are several requirements that must be met in order to make use of the IPP environment:

1. The programmer must have access to a Sun 3 workstation. It should be noted that a Sun 3 has color capability and this capability is used when displaying the images. Hence, a Sun 2 workstation can not be used. It should also be noted that the executable code provided with this package will not run on a Sun 2 even if the display routine is not used. This is due to the fact that Sun 2 workstations use a Motorola 68010 microprocessor and the Sun 3 workstations use a Motorola 68020. If use of a Sun 2 is necessary, all code except the image display routine can be recompiled and run on the Sun 2.

2. Access to the iPSC Vector cube is also required to use the IPP environment. Before an application can be executed the user must login to the hypercube from a window of suntools and execute a program to allow the software running on the Sun workstation to communicate with the hypercube. This procedure is outlined in detail later in this manual.

3. The user's application must be run from within the "suntools" environment on the Sun workstation to use any image display routines.

4. The appropriate IPP environment files for the Sun and the iPSC hypercube must be available in the user's directories as specified on the disk installation instructions.

Example Program

A few example programs have been provided to demonstrate the capabilities of the IPP environment and to serve as programming examples to aid in the development of other applications. The first is a simple program that computes the Fast Fourier Transform of an image specified by the user and displays the result. The program is listed below and has been provided with the example source files:

```
/* fftdemo program */
#include <stdio.h>
#include "ipp.h"

main(argc,argv)
int argc;
char *argv[];

{
/*******************************/
/* the image processor identifier */
IP_TYPE ip;

/* the image buffer */
I_TYPE image[X_PIX*Y_PIX];

/* the histogram buffer */
H_TYPE h[256];

/* misc. buffer */
G_TYPE g[256];

FILE *fp;
int i,j,k;
short bit;

/*******************************/

if (argc==1) {
   printf("No file \n");
   exit();
}
```

```
if ((fp=fopen(*++argv,"r")) == (FILE *)NULL) {
   printf("File not found\n");
   exit(-1);
}

/*******************************/
printf("starting   \n");

/* start an image processor */
ip=start_ip();

/* initialize suncore */
sun_init();

/* copy the image to the I array */
ftoI(ip,fp);

/* copy the I array to the A array */
ItoA(ip);

/* save a copy in B */
AtoB(ip);

/* get a copy of the image back from I */
get_I(ip,image);

/* display the buffer 'image' */
show(image,0);

/* do the fft on A */
fft(ip);

/* take the magnitude */
mag(ip);

/* put the dc frequency terms in the middle */
swapq(ip);

/* scale to do the log transform */
g[0]=1.0;
g[2]=255.0;
put_G(ip,g,2);
scale(ip);

/* do the log transform */
log(ip);

/* scale so we can convert to 8 bit integers in I */
g[0]=0.0;
g[2]=255.0;
put_G(ip,g,2);
scale(ip);
```

```
    /* convert A to 8 bit I array */
    AtoI(ip);

    /* get a copy of I */
    get_I(ip,image);

    /* show the result */
    show(image,0);

    /* terminate suncore */
    sun_term();

    }
```

The first step tc using this program is to enter the Suntools environment. This is done by typing

```
suntools
```

More information about the suntools environment can be obtained by access the Sun online system manual. This is done by typing

```
man suntools
```

More information on the man command is made available by typing

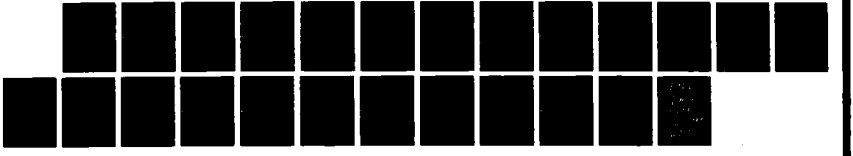
```
man man
```

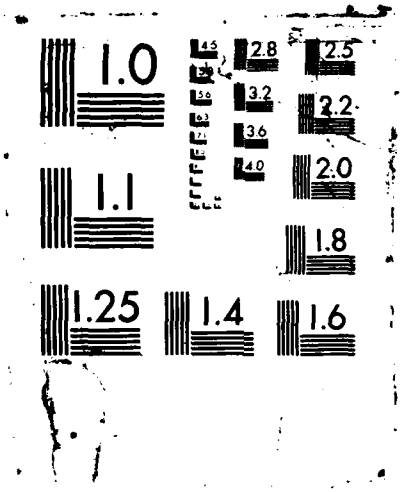
To compile the program the following files must be available on the Sun workstation.

```
ipp.h
ipplib.a
cubelibv.a
makefile
host.c
```

AD-A192 033 IMAGE PROCESSING USING A PARALLEL ARCHITECTURE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING B R HODGES DEC 87 AFIT/GE/ENG/87D-25 2/2

UNCLASSIFIED F/G 12/9 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

where host.c is the fftdemo program. To compile the program type

```
make host
```

After the file is compiled with no errors, the communications link between the hypercube and the Sun workstation must be established. Two "windows" are required to do this: one for the hypercube and one for the application running on the Sun. From one of two windows type

```
rlogin vect_cube
```

This window will be referred to as the hypercube window. After logging into the vector cube, from the hypercube window type

```
ipp
```

This program will first attempt to obtain ownership of the hypercube. This is necessary since only one user is allowed to use the cube at one time. The program will not proceed until it has obtained ownership of the cube. Once the "DONE" message is printed you will be instructed to run your application. If for any reason the Ethernet connection is lost before the application program exits normally, move back to the hypercube window and type

```
restart
```

As your application is running, any error messages generated

by the cube will appear in the hypercube window. By moving the cursor from this window to the second window you be will be able to run the fft demo program. From the second window type

```
host image.img
```

where "image.img" is a 64K image file. The image will be read in, and the fast fourier transform computed. Next, the image will be scaled and translated for display. Finally the image will be displayed on the screen until one of the three mouse buttons is pressed while the cursor is in the applications window. If the cursor is moved from the application window during the display of an image, the suntools environment will change the screen colors. The image can be viewed with its proper shading by moving the cursor back into the application window.

Image Processor Program Routines

This section presents each of the C routines written for the host. An explanation of the purpose and possible uses of the function call will be presented and a code section will demonstrate the proper syntax for using the function. The file ipp.h contains the "define" statements for the variables required for the number of pixels and the number of nodes.

The IPP set of routines makes available a unique working environment for an image processing applications programmer. The application can start an image processor program to

perform a variety of operations on an image. Once an image processor program is started it will exist until the application has completed running and will maintain any image until the image is overwritten by a new image or it is explicitly erased.

In each image processor program there are five data arrays. The integer I array is used to feed images to the image processor program and operations that apply to integer image data will be performed on this array. The next two arrays, A and B, are complex data arrays that are used for operations that require real or complex quantities. These arrays can not be loaded directly, but information can be converted to and from the I array. The last two arrays, H and G, are smaller general use arrays. The H array is a 256 element integer array used mainly for histogram related operations. The G array is floating point valued and is used for a number of purposes.

The start_ip function is used to start an image processor to perform image processing tasks. An image processor program identifier is returned and used to access the image processor program in subsequent function calls. This function can be called multiple times to create multiple image processor programs.

```
IP_TYPE my_ip1, my_ip2;
      .
      .
my_ip1 = start_ip();
      .
```

```
        .
my_ip2 = start_ip();
        .
        .
```

The put_I function is used to load the I data array with an image. This must be done after the start_ip routine but before any image operations can be performed.

```
IP_TYPE my_ip;
IMAGE_TYPE my_image[X_PIX*Y_PIX];
        .
        .
put_I(my_ip,my_image);
        .
        .
```

The get_I function is used to retrieve the I array and store it in an image array. Using the same variables as in load_I the function call is

```
get_I(my_ip,my_image)
```

The get_H function is used to retrieve the results of histogram type operations. The H array is a 256 integer long array.

```
integer my_histogram[256];
        .
        .
get_H(my_ip, my_histogram);
        .
        .
```

The put_H function is used to store data in the H array. An example of the use of this function is when performing the remapping of an image during histogram equalization.

```
put_H(my_ip, my_histogram);
```

The hist function determines the histogram of the image in I and stores it in H. The histogram can then be retrieved by using get_H.

```
hist(my_ip);
```

The map function maps the intensity values of the image I into new values in the H array. Thus, if H[0]=5 then all pixels in I whose value is 0 will be given a new value of 5. This function is used to remap images after their histogram has been equalized in the hist_equ function.

```
map(my_ip);
```

The hist_equ function performs a histogram equalization on the image in I. It gets the histogram of the image, equalizes the histogram, and remaps the image based on this new histogram.

```
hist_equ(my_ip);
```

Many image processing operations require real or complex values to perform the operation. Two complex arrays A and B are provided in the image processor program to do these calculations. The following functions are provided to convert between image data and complex data.

```
ItoA(my_ip);        /* convert integer array I to */
                    /* complex array A            */
```

```
AtoI(my_ip);          /* convert real part of complex */
                      /* array A to integer array I   */
```

There are three functions that move data between the A and B complex arrays. They move data from A to B, B to A, or exchange the data between A and B.

```
AtoB(my_ip);      /* copy complex array A to B */

BtoA(my_ip);      /* copy complex array B to A */

AexB(my_ip);      /* exchange the complex arrays */
```

Some of the operations that are possible for the A array are

```
zero(my_ip);      /* zeroing the A array */

range(my_ip);     /* find the range of the A array */
                  /* and store the results in      */
                  /* G[0] and G[1].                */

mag(my_ip);       /* taking the magnitude of A */

phase(my_ip);     /* taking the phase of A */

conjugate(my_ip); /* taking the complex conjugate */
                  /* of the A array               */

transpose(my_ip); /* transposing A */

fft(my_ip);       /* taking the fft of A */

ifft(my_ip);      /* taking the inverse fft of A */
```

The first element of the G array is used to allow operating on the A array with a constant.

```
Cmult(my_ip);     /* multiply A by G[0] */

Cadd(my_ip);      /* add G[0] to A */
```

There are also several functions that perform operations that involve both the A and B complex arrays. For all these operations the result is stored in A. They are

```
add(my_ip);      /* add B to A */

sub(my_ip);      /* subtract B from A */

mult(my_ip);     /* multiply A by B */

div(my_ip);      /* divide A by B */

mt(my_ip);       /* perform the matching transform */

conv(my_ip);     /* perform the circular convolution */
                 /* of A and B                       */

corr(my_ip);     /* perform the correlation of */
                 /* A and B                    */

op3(my_ip);      /* perform the convolution of A */
                 /* with the 3x3 operator stored */
                 /* in the first 9 elements of G */

op5(my_ip);      /* perform the convolution of A  */
                 /* with the 5x5 operator stored  */
                 /* in the first 25 elements of G */

op7(my_ip);      /* perform the convolution of A  */
                 /* with the 7x7 operator stored  */
                 /* in the first 49 elements of G */

scale(my_ip);    /* scale the A array to be in the */
                 /* range specified by the real    */
                 /* parts of G[0] to G[1]          */
```

A variable bandpass filter is provided for frequency domain filtering. The frequency range is considered to exist in a normalized range 0.0 to 1.0 with 0.0 being the lower end of the spectrum. Three parameters, in the range 0.0 to 1.0, that specify the filter characteristics are stored in the real part of G[0], G[1], and G[2]. The first and second

indicate the bandpass region and the third indicate the width of a roll-off region to either side of the pass band. For example, to specify a low pass filter with a sharp cut-off the parameters would be 0.0, 0.5, and 0.0. A high pass filter would be 0.5, 1.0, and 0.0. Finally, a bandpass filter with a gradual roll-off might be 0.23, 0.77, and 0.1. Once the filter parameters have been stored in G, the function call is

```
filter(my_ip);
```

Three routines are provided to display an image array on the Sun workstation.

```
sun_init();     /* initializes the image display */
                /* capabilities                  */

show(image,color); /* displays the image in the */
                   /* buffer "image" color=0,   */
                   /* gives grey scales.        */
                   /* color=1, color display    */

term_sun();     /* terminate the image display   */
                /* capabilities                  */
```

The sun_init() routine must be used called before any images can be displayed. The show() routine will display an image until one of the three mouse buttons is pressed. The term_sun() routine should be called before the application routine exits back to the system or when the image display capabilities are no longer needed.

When displaying the frequency domain representation of an image, the low frequency terms are in the four corners. A

routine that swaps the diagonal quadrants (1 with 3 and 2 with 4) of an image is provided to arrange the A array such that low frequency terms are in the middle. The filter function can be used on this representation but, low frequency terms are near 1.0 instead of near 0.0. The function call is

```
swap_quads(my_ip);
```

An Application

A simple menu-driven image processing program has been provided for use as an example or as an interactive image processing tool. The entire source code for the program is provided on disk. The executable program is included along with the source code so it is not necessary to compile the program. This program must be run from a "gfxtool" suntool's window to allow the text to be displayed. After establishing the communications as outlined in the example program section, type

```
ippdemo  image.img
```

in the gfxtool window where image.img is a 64K image file. An menu will present the following options:

1. Pick another image file
2. Low-pass filter the image
3. High-pass the image
4. Bandpass the image
5. Show the FFT of the image
6. Histogram Equalize the image
7. Convolve a 3x3 operator with the image
8. Convolve two images
9. Correlate two images
10. Matching Transform
11. Save Image

These options represent the some of the IPP environment routines available for the programmer. Of course, these routines can be used with whatever routines have been written by the user in an applications program to perform variety of tasks.

Software Expansion

This section will outline the addition of a routine to the set of existing routines. As an example, a routine already in the software will be selected and the step by step procedure taken to add it to the software will be retraced. This discussion assumes the source files, from the floppy disk, have been installed per the instructions in the README.IPP file on the floppy disk. It is suggested that the programmer become familiar with the source code files by

reading the documentation in each file before proceeding in this explaination.

The routine AtoB, which copies array A into array B was selected because it requires no internode communications. Once this routine is understood, more complicated routines can be added.

Adding this routine requires modification to both the host program and the node program. The first step in modifying the host software is to add an entry in the ipp.h file on the Sun that represents the command code. This code is used by the node program to determine what action is to be taken. The command code for AtoB in the file ipp.h is "defined" as 26.

```
#define AtoB_CMD  26+IPP
```

where IPP is a constant used to offset command values so that these commands will not interfere with other programs that could possibly be written to run on the system at the same time. It should be noted that all command codes must be unique. The next step is writing the driver C function that will be called to issue the AtoB command. This function is quite simply

```
#include "ipp.h"

AtoB(ip)
IP_TYPE ip;

{
  put_cmd(ip,AtoB_CMD);
}
```

This program sends the command to the image processor selected by the variable ip. Of course, the AtoB command could be expanded to include any of the other calls defined in the previous section.

Next, the AtoB file must be included in the makefile for compilation. Type

```
man makefile
```

on the Sun for additional information. Once the "AtoB.o" entry has been added to the makefile, type

```
make lib
```

This will produce a new ipplib.a file to replace the old ipplib.a file. The host software modifications are now complete.

The first step in modifying the node software is to add the code that will interpret the AtoB command. In the file node.f on the iPSC add a new "else if" clause to the Fortran program.

```
      .
      .
else if (cmd .eq. 26) then
  call AtoB(A,B)
      .
      .
```

Where AtoB is the Fortran program on the iPSC that actually does the work. This program is in the file AtoB.f and is

```
subroutine AtoB(A,B)
include 'ippcomm.h'

real A(MM,N), B(MM,N)

call scopy(2*L,A,1,B,1)

return
end
```

The file ippcomm.h is a file for the Fortran node programs that defines the variables that relate to image size (MM, N, and L). The scopy subroutine is an iPSC Vector Library call documented in the iPSC Program Development Guide. Finally, an entry for this new Fortran subroutine must be added to the makefile on the iPSC. Once this is done type

```
make ippnode
```

and a new node program to replace the old ippnode file will be generated.

Once these steps are accomplished, host programs can be written using the new AtoB routine.

Bibliography

1. Almasi, George and Stephen Harvey. "An Introduction to Parallel Processing," Journal of Electronic Defense, 9: 31-42 (May 1986)

2. Array Processing and Digital Signal Processing. Product Information. Advanced Micro Devices, Sunnyvale, 1986.

3. Baer, Jean-Loup. Computer Systems Architecture. Rockville, Maryland: Computer Science Press, 1980.

4. Booch, Grady. Software Engineering with Ada. Menlo Park, California: The Benjamin/Cummings Publishing Company, 1983.

5. Brigham, Oran E. The Fast Fourier Transform. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1974

6. Cooley, J.W. and J.W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, 19: 297-301 (April 1965).

7. De Fatta, Richard P. Target Recognition Using Three Dimensional Laser Range Imagery, MS Thesis AFIT/GEP/ENP/86D-2. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986 (AD-A177-755).

8. Fleury, Peter A. "Covert Penetration Systems: Future Strategic Aircraft Missions Will Require a New Sensor System Approach," Proceedings of NAECON 1:220-226 (May 1986).

9. Frederickson, Paul O. (Computing and Communication Division, Los Alamos National Laboratory). Scientific Computation on Highly Parallel Supercomputers. Briefing at the Massively Parallel Supercomputer Seminar in Dayton Ohio, 22 July 1987.

10. Hall, Ernest L. Computer Image Processing and Recognition. New York: Academic Press, 1979.

11. Harris, D.B. et al. "Vector Radix Fast Fourier Transform," IEEE Internat. Conf. on Acoust., Speech, Signal Process. Rec.: 548-551 (1977).

12. Hollingum, Jack. Machine Vision: The Eyes of Automation. Berlin: IFS (Publications) Ltd., 1984.

13. ITEX-100 Programmer's Manual. Part # 47-S10008-02. Imaging Technology Inc. Woburn Massachusetts

14. Levine, Martin D. Vision in Man and Machine. New York: McGraw-Hill Book Co., 1985.

15. Nussbaumer, H.J. Fast Fourier Transform and Convolution Algorithms. New York: Springer-Velag, 1982.

16. Oppenheim, Alan V. and Ronald W. Schafer. Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1975.

17. Palmer, J.F. "Hypercube Algoritms," Proceedings of the 1985 ASME International Computers in Engineering Conference. Volume 3. The American Society of Mechanical Engineers, New York, August 1985.

18. Pease, Marshall C. "An Adaptation of the Fast Fourier Transform for Parallel Processing," Journal of the Association for Computing Machinery, 15: 252-264 (April 1968).

19. Pratt, William K. Digital Image Processing. New York: John Wiley & Sons, 1978.

20. Pressman, Roger S., Software Engineering: A Practitioner's Approach. New York: McGraw-Hill Book Co., 1982.

21. Quinn, Micheal J. Designing Efficient Algorithms for Parallel Computers. New York: McGraw-Hill Book Co., 1987.

22. Ratkovic, J.A. Structuring the Components of the Image Matching Problem. Interim Report contract F49620-77-C-0023, December 1979 (AD-A082-085).

23. Stanley, William D. "Fast Fourier Transforms on Your Home Computer," Byte, 3: 14-25 (December 1978). McGraw-Hill, 1985

24. Trussell, H. Joel. "Processing of X-ray Images," Proceedings of the IEEE, 69: 615-627 (May 1981).

25. Walton, Stephen R. "Fast Fourier Transforms on the Hypercube," Presented at the Second Conference on Hypercube Multiprocessors, 1986.

26. Whitten, Gary et al. "Temperature-driven Segmentation for Autonomous Anit-tank Weapons," Proceedings of NAECON 1:186-196 (May 1986).

27. Wiley, Paul. "A Parallel Architecture Comes of Age at Last," IEEE Spectrum, 24: 46-50 (June 1987).

28. Woolett, Jerry F. "MMWR/FLIR/ATR Fusion Proof of Concept," Proceedings of NAECON 1:180-185 (May 1986).

29. Ziemer, R.E. and W.H. Tranter. Principles of Communications. Boston: Houghton Mifflin Co., 1976.

Additional References

Stremler, Ferrel G. Introduction to Communications Systems. Reading, Massachusetts: Addison-Wesley Publishing Co., 1982.

Kuck, David J. et al. High Speed Computer and Algorithm Organization. New York: Academic Press, Inc., 1977.

Baase, Sara, Computer Algorithms: Introduction to Design and Analysis. Reading, Massachusetts: Addison-Wesley Publishing Co., 1978.

Blahut, Richard E., Fast Algorithms for Digital Signal Processing. Reading, Massachusetts: Addison-Wesley Publishing Co., 1985.

Ludeman, Lonnie C. Fundamentals of Digital Signal Processing. New York: Harper & Row, 1986.

Bose, N.K. Digital Filters. New York: North-Holland, 1985.

The FPS T Series Parallel Supercomputer. Floating Point Systems, Briefing at the Massively Parallel Supercomputer Seminar in Dayton Ohio, 22 July 1987.

Rogstad, D.H. (Caltech/JPL). JPL Hypercube Project. Briefing at the Massively Parallel Supercomputer Seminar in Dayton Ohio, 22 July 1987.

Directions in Scientific Computing. Floating Point Systems, Briefing at the Massively Parallel Supercomputer Seminar in Dayton Ohio, 22 July 1987.

George, Mike (Northrop Aircraft Division). Parallel Processing and Computational Physics. Briefing at the Massively Parallel Supercomputer Seminar in Dayton Ohio, 22 July 1987.

Lewis, Donald E. Two Dimensional Fast Fourier Transforms in Image Processing. Technical Report AFAMRL-TR-84-006. AMD, AFSC, Wright-Patterson AFB OH, January 1984 (AD-A139-997).

Hicks, Robert C. Comparion of Arithmetic Requirements for the PFA, WFTA, SWIFT, MFFT, FFT, and DFT Algorithms. Technical Report RE-93-6. U.S. Army Missle Command, Redstone Arsenal, Alabama, November 1982.

Adam, John A. "Counting the Weapons," IEEE Spectrum, 23: 46-56 (July 1986).

Schindler, Max. "Parallel Processing," Electronic Design, 35:91-100 (January 1987).

Callaghan, Tom. "Applications of Digital Signal Processing," Defense Science and Electronics, 4: 52-60.

Jain, Anil K. "Advances in Mathematical Models for Image Processing," Proceedings of the IEEE, 69: 502-528 (May 1981).

Oppenheim, Alan V. Jae S. Lim. "The Importance of Phase in Signals," Proceedings of the IEEE, 69: 529-541 (May 1981).

Barrow, Harry G. and Jay M. Tenebaum. "Computational Vision," Proceedings of the IEEE, 69: 572-595 (May 1981).

Rosenfeld, Azriel. "Image Pattern Recognition," Proceedings of the IEEE, 69: 596-614 (May 1981).

Landgrebe, David A. "Analysis Technology for Land Remote Sensing," Proceedings of the IEEE, 69: 628-642 (May 1981).

Woods, R.E. and R.C. Gonzalez. "Real-Time Digital Image Enhancement," Proceedings of the IEEE, 69: 572-595 (May 1981).

Luetkemeyer, Kent. "Evaluation of Segmentation Techniques Applied to Prescreened Areas of Multi-Sensor Imagery," Proceedings of NAECON 1:197-204 (May 1986).

Woolfson, Martin G. "Classifier Integration with Multiple Sensors," Proceedings of NAECON 1:205-209 (May 1986).

Kroupa, Richard F. and Bruce J. Schachter. "ATR Development at Westinghouse," Proceedings of NAECON 1:210-214 (May 1986).

Gibson, Laurie and Dean Lucas. "Pyramid Algorithms for Automated Target Recongnition," Proceedings of NAECON 1:215-219 (May 1986).

Brass, A. et al. "Two and Three Dimensional FFTs on Highly Parallel Computers," Parallel Computing, 3: 167-184 (1986).

McLachlan, Dan Jr. "The Role of Optics in Applying Correlation Functions to Pattern Recognition," Journal of the Optical Society of America, 52: 454-459 (April 1962).

Abu-Mostafa, Yaser S. and Demetri Psaltis. "Optical Neural Computers," Scientific American, 256: 88-95 (March 1987).

Mueller, Philip T et al. "Parallel Algorithms for the Two-Dimensional FFT," Proceedings of the Conference on Pattern Recognition, 1: 497-502 (December 1980).

Blinchikoff, Herman J. and Anatol I. Zverev. Filtering in the Time and Frequency Domains. New York: John Wiley & Sons, 1976.

Fukunaga, Keinosuke. Introduction to Statistical Pattern Recongnition. New York: Academic Press, 1972.

Bronson, Richard. Matrix Methds: An Introduction. New York: Academic Press, 1970.

Taylor, Fred and Steve L. Smith. Digital Signal Processing in FORTRAN. Lexington, Massachusetts: Lexington Books, 1976.

Pavlidis, Theo. Algorithms for Graphics and Image Processing. Rockville, Maryland: Computer Science Press, 1982.

Norton, Alan and Allan J. Silberger. "Parallelization and Performance Analysis of the Cooley-Tukey FFT Algorithm for Shared-Memory Architectures," IEEE Transactions on Computers, 36: 581-591 (May 1987).

Magee, Micheal J. et al. "Experiments in Intensity Guided Range Sensing Recognition of Three-Dimensional Objects," IEEE Transactions on Patteru Analysis and Machine Intelligence, 7: 629-637 (November 1985).

Machuca, Raul and Alton L. Gilbert. "Finding Edges in Noisy Scenes," IEEE Transactions on Pattern Analysis and Machine Intelligence, 3: 103-111 (January 1981).

Berger, Marsha J. and Shahid H. Bokhari. "A Partitioning Stategy for Nonuniform Problems on Multiprocessors," IEEE Transactions on Computers, 5: 570-579.

Preparata, Franco P. and Jean Vuillemin. "The Cube-Connected-Cycles: A Versatile Network for Parallel Computation," Joint Services Electronics Program contract N00014-79-C-0424 (AD-085-846).

Therrien, Charles W. et al. "A Multiprocessor System for Simulation of Multisensor Distributed Decision Algorithms," Asilomar Conference on Circuits, Systems & Computers. IEEE Computer Society, Washington, November 1985.

Ramanamurthy, D.V. et al. "Parallel Algorithms for Low Level Vision on the Homogeneous Multiprocessor," Proceedings of the Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, Washingon, June 1986.

Bhanu, Bir. "Automatic Target Recognition: State of the Art Survey," IEEE Transactions on Aerospace and Electronic Systems, 22: 364-379 (July 1986).

Duff, M.J.B. S. Levialdi (editors). Languages and Architecutures for Image Processing. London: Academic Press, 1981.

Mitra, Sanjit K. and Michael P. Ekstrom (editors). Two-Dimensional Digital Signal Processing. Stroudsburg, Pennsylvania: Halsted Press, 1978.

Onoe, Morio et al (editors). Real-Time/Parallel Computing: Image Analysis. New York: Plenum Press, 1981.

Simon, J.C. and R.M. Haralick (editors). Digital Image Processing. Boston: D. Reidel Publishing Co., 1981.

VITA

Captain Billy R. Hodges was born on 5 October 1961 in Roswell, New Mexico. He graduated from high school in Memphis, Tennessee, in 1980 and attended Memphis State University, from which he received the degree of Bachelor of Science in Electrical Engineering in May 1983. Upon graduation he entered Officer Training School where he received a commission in the USAF. His first assignment, at Aeronautical Systems Division, Wright-Patterson AFB, began in September 1983. He served as an avionics engineer in support of the Ground Collision Avoidance System program for fighter aircraft until entering the School of Engineering, Air Force Institute of Technology, in June 1986.

Permanent address:

3048 Ruffle Drive
Bartlett, TN 38134

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/ENG/87D-25	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code): Air Force Institute of Technology (AU), Wright-Patterson AFB, OH 43433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
IMAGE PROCESSING USING A PARALLEL ARCHITECTURE

12. PERSONAL AUTHOR(S)
Billy R. Hodges, Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1987 December	117

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	09		Image Processing, Convolution, Parallel Processing, Correlation
12	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: IMAGE PROCESSING USING A PARALLEL ARCHITECTURE

Thesis Advisor: Walter D. Seward, Lt Col, USAF
Associate Professor of Electrical Engineering

Approved for public release: IAW AFR 190-1.
14 May 87
[illegible] Development
Air Force Institute
Wright-Patterson

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Walter D. Seward, Lt Col, USAF	513-255-2024	AFIT/ENG

DD Form 1473, JUN 86 — Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

This study developed a set of low level image processing tools on a parallel computer that allows concurrent processing of images in order to support development of systems that use multiple images to gather information. The parallel computer used is a collection of powerful processors connected in a hypercube topology.

The software developed simplifies the interface between the parallel computer and the applications developer by providing a library of functions in the C programm' g language. These functions are used to control "image processor programs" that run independently in the parallel computer and perform the image processing operations. The complexities of the parallel processor are hidden and replaced with a flexible structure specifically designed for image processing. This structure provides a simplified interface, but also acts as a framework to which additional image processing operations can be added.

Timing measurements indicate that, in addition to providing a unique applications development environment, the set of tools offers a significant reduction in the time required to perform some commonly used image processing operations.

END

DATE

FILMED

6-1988

DTIC